

Trimming and gluing Gray codes

Petr Gregor

Department of Theoretical Computer Science
and Mathematical Logic
Charles University
11800 Praha 1, Czech Republic
gregor@ktiml.mff.cuni.cz

Torsten Mütze

Institut für Mathematik
TU Berlin
10623 Berlin, Germany
muetze@math.tu-berlin.de

ABSTRACT. We consider the algorithmic problem of generating each subset of $[n] := \{1, 2, \dots, n\}$ whose size is in some interval $[k, l]$, $0 \leq k \leq l \leq n$, exactly once (cyclically) by repeatedly adding or removing a single element, or by exchanging a single element. For $k = 0$ and $l = n$ this is the classical problem of generating all 2^n subsets of $[n]$ by element additions/removals, and for $k = l$ this is the classical problem of generating all $\binom{n}{k}$ subsets of $[n]$ by element exchanges. We prove the existence of such cyclic minimum-change enumerations for a large range of values n , k , and l , improving upon and generalizing several previous results. For all these existential results we provide optimal algorithms to compute the corresponding Gray codes in constant time $\mathcal{O}(1)$ per generated set and space $\mathcal{O}(n)$. Rephrased in terms of graph theory, our results establish the existence of (almost) Hamilton cycles in the subgraph of the n -dimensional cube Q_n induced by all levels $[k, l]$. We reduce all remaining open cases to a generalized version of the middle levels conjecture, which asserts that the subgraph of Q_{2k+1} induced by all levels $[k - c, k + 1 + c]$, $c \in \{0, 1, \dots, k\}$, has a Hamilton cycle. We also prove an approximate version of this conjecture, showing that this graph has a cycle that visits a $(1 - o(1))$ -fraction of all vertices.

KEYWORDS: Gray code, subset, combination, loopless algorithm, hypercube

1. INTRODUCTION

Generating all objects in a combinatorial class such as permutations, subsets, combinations, partitions, trees, strings etc. is one of the oldest and most fundamental algorithmic problems, and such generation algorithms appear as core building blocks in a wide range of practical applications (see the survey [Sav97]). In fact, half of the most recent volume [Knu11] of Donald Knuth's seminal series *The Art of Computer Programming* is devoted entirely to this fundamental subject. The ultimate goal for algorithms that efficiently generate each object of a particular combinatorial class exactly once is to generate each new object in constant time, which is best possible. Such optimal algorithms are sometimes called *loopless algorithms*, a term coined by Ehrlich in his influential paper [Ehr73]. Note that a constant-time algorithm requires in particular that consecutively generated objects differ only in a constant amount, e.g., in a single transposition of a permutation, in adding or removing a single element from a set, or in a single tree rotation operation. These types of orderings have become known as *combinatorial Gray codes*. Here are two fundamental examples for this kind of generation problems: (1) The so-called *reflected Gray code* is a method to generate all 2^n many subsets of $[n] := \{1, 2, \dots, n\}$ by repeatedly adding or removing a single element. It is named after Frank Gray, a physicist and researcher at Bell Labs, and appears in his patent [Gra]. The reflected Gray code has many interesting properties (see [Knu11, Section 7.2.1.1]), and there is a simple loopless algorithm to compute it [Ehr73, BER76]. (2) Of similar importance in practice is the problem of generating all $\binom{n}{k}$ many k -element subsets of $[n]$ by repeatedly exchanging a single element. Also for this problem, loopless algorithms are well-known [TL73, Ehr73, BER76, EM84, EHR84, Rus88, Cha89, JM95] (see also [Knu11, Section 7.2.1.3]).

In this work we consider far-ranging generalizations of the classical problems (1) and (2). Specifically, we consider the algorithmic problem of generating all (or almost all) subsets of $[n]$ whose size is in some interval $[k, l]$, $0 \leq k \leq l \leq n$, by repeatedly adding or removing a single element, or by exchanging a single element (this will be made more precise in a moment). We recover the classical problems (1) and (2) mentioned before as special cases by setting $k = 0$ and $l = n$, or by setting $k = l$, respectively. It turns out that the entire parameter range in between these special cases offers plenty of room for surprising discoveries and hard research problems (take a peek at the known results in Figure 1 below).

In a computer a subset of $[n]$ is conveniently represented by the corresponding characteristic bitstring x of length n , where all the 1-bits of x correspond to the elements contained in the set, and the 0-bits to the elements not contained in the set. The before-mentioned subset generation problems can thus be rephrased as Hamilton cycle problems in subgraphs of the *cube* Q_n , the graph that has as vertices all bitstrings of length n , with an edge between any two vertices (=bitstrings) that differ in exactly one bit. We refer to the number of 1-bits in a bitstring x as the *weight* of x , and we refer to the vertices of Q_n with weight k as the *k-th level* of Q_n (there are $\binom{n}{k}$ vertices on level k). Moreover, we let $Q_{n,[k,l]}$, $0 \leq k \leq l \leq n$, denote the subgraph of Q_n induced by all levels $[k, l]$. In terms of sets, the vertices of the cube Q_n correspond to subsets of $[n]$, and flipping a bit along an edge corresponds to adding or removing a single element. The weight of a bitstring corresponds to the size of the set, and the vertices on level k correspond to all k -element subsets of $[n]$.

One of the hard instances of the before-mentioned general enumeration problem in $Q_{n,[k,l]}$ is when $n = 2k + 1$ and $l = k + 1$. The existence of a Hamilton cycle in the graph $Q_{2k+1,[k,k+1]}$ for any $k \geq 1$ is asserted by the well-known *middle levels conjecture*, raised independently in the 80's by Havel [Hav83] and Buck and Wiedemann [BW84]. The conjecture has also been attributed to Dejter, Erdős, Trotter [KT88] and various others, and also appears in the popular books [Win04, Knu11, DG12]. The middle levels conjecture has attracted considerable attention over the last 30 years [Sav93, FT95, SW95, Joh04, DSW88, KT88, DKS94, HKRR05, GŠ10, MW12, SSS09, SA11], and a positive solution, i.e., an existence proof for a Hamilton cycle in $Q_{2k+1,[k,k+1]}$ for any $k \geq 1$, has been announced only recently.

Theorem 1 ([Müt16]). *For any $k \geq 1$, the graph $Q_{2k+1,[k,k+1]}$ has a Hamilton cycle.*

The following generalization of the middle levels conjecture was proposed in [GŠ10].

Conjecture 2 ([GŠ10]). *For any $k \geq 1$ and $c \in \{0, 1, \dots, k\}$, the graph $Q_{2k+1,[k-c,k+1+c]}$ has a Hamilton cycle.*

Conjecture 2 clearly holds for all $k \geq 1$ and $c = k$ as $Q_{2k+1,[0,2k+1]} = Q_{2k+1}$ (this is problem (1) from before). It is known that the conjecture also holds for all $k \geq 1$ and $c = k - 1$ [EHH01, LS03] and $c = k - 2$ [GŠ10]. By Theorem 1 we know that it also holds for all $k \geq 1$ and $c = 0$. As far as small cases are concerned, computer experiments show that $Q_{2k+1,[k-c,k+1+c]}$ indeed has a Hamilton cycle for all $k \leq 6$ and all relevant values of c (the biggest instance in this range not yet covered by the before-mentioned general results is $Q_{13,[3,10]}$ with 8008 vertices).

Another generalization of Theorem 1 in a slightly different direction (still a special case in our general framework) is the following result.

Theorem 3 ([MS15]). *For any $n \geq 3$ and $k \in \{1, 2, \dots, n - 2\}$, the graph $Q_{n,[k,k+1]}$ has a cycle that visits all vertices in the smaller bipartite class.*

The idea for the proof of Theorem 3 (induction over n) was first presented in [Hav83]. In that paper, the theorem was essentially proved conditional on the validity of the hardest case $n = 2k + 1$, the middle levels conjecture, which was established as a theorem (Theorem 1) only much later. In

[MS15], Theorem 3 was proved unconditionally, and the proof technique was refined further to also prove Hamiltonicity results for the so-called bipartite Kneser graphs, another generalization of the middle levels conjecture.

Conjecture 2 and Theorem 3 immediately suggest the following common generalization: For which intervals $[k, l]$ does the cube $Q_{n,[k,l]}$ have a Hamilton cycle? The graph $Q_{n,[k,l]}$ is bipartite (the two partition classes are given by the parity of the number of 1-bits of the vertices), and it is clear that a Hamilton cycle can exist only if the two partition classes have the same size, which happens only for odd dimension n and between two symmetric levels k and $l = n - k$ (Conjecture 2). However, we may slightly relax this question, and ask for a long cycle. To this end, we denote for any bipartite graph G by $v(G)$ the number of vertices of G , and by $\delta(G)$ the difference between the larger and the smaller partition class. Note that in any bipartite graph G (as $Q_{n,[k,l]}$) the length of any cycle is at most $v(G) - \delta(G)$ (i.e., the cycle visits all vertices in the smaller partition class). We call such a cycle a *saturating cycle*. Observe that if both partition classes have the same size (i.e. $\delta(G) = 0$), then a saturating cycle is a Hamilton cycle. Hence saturating cycles naturally generalize Hamilton cycles for unbalanced bipartite graphs. The right common generalization of Conjecture 2 and Theorem 3 therefore is:

Question 4. For which intervals $[k, l]$ does the cube $Q_{n,[k,l]}$ have a saturating cycle?

A saturating cycle necessarily omits some vertices (exactly $\delta(Q_{n,[k,l]})$ many) from the larger bipartite class. However, if we insist on all vertices of $Q_{n,[k,l]}$ to be included in a cycle, then this can be achieved by allowing steps where instead of only a single bitflip, two bits are flipped (the underlying graph $Q_{n,[k,l]}$ is augmented by adding distance-2 edges). In this case we may ask for a (cyclic) enumeration of all vertices of $Q_{n,[k,l]}$ that minimizes the number of these ‘cheating’ distance-2 steps, i.e., for an enumeration that has only $\delta(Q_{n,[k,l]})$ many distance-2 steps (this is clearly the least number one can hope for). We call such an enumeration a *tight enumeration*. A tight enumeration can be seen as a travelling salesman tour of length $v(Q_{n,[k,l]}) + \delta(Q_{n,[k,l]})$ through all vertices of $Q_{n,[k,l]}$, where distances are measured by Hamming distance (=number of bitflips). We may ask in full generality:

Question 5. For which intervals $[k, l]$ is there a tight enumeration of the vertices of $Q_{n,[k,l]}$?

Observe that if both partition classes of $Q_{n,[k,l]}$ have the same size (i.e. $\delta(Q_{n,[k,l]}) = 0$), then a tight enumeration is a Hamilton cycle in this graph. Note also that Question 5 is a sweeping generalization of the following well-known result regarding problem (2) mentioned before, first proved in [TL73].

Theorem 6 ([TL73]). *For any $n \geq 2$ and $1 \leq k \leq n - 1$ there is a cyclic enumeration of all weight k bitstrings of length n such that any two consecutive bitstrings differ in exactly 2 bits.*

In fact, several of the subsequently mentioned results of this paper will be proved by extending the original approach from [TL73] to prove Theorem 6.

1.1. Our results. In this work we answer Question 4 and Question 5 for a large range of values n , k and l . The different ranges of parameters covered by our results are illustrated in Figure 1. Moreover, we provide optimal algorithms to compute the corresponding saturating cycles/tight enumerations.

Our first set of results resolves Question 4 positively for all possible values of k and l except the cases covered by Conjecture 2 (the case $l = k + 1$ is already covered by Theorem 3), see the left-hand side of Figure 1.

Theorem 7. *For any $n \geq 3$ the graph $Q_{n,[k,l]}$ has a saturating cycle in the following cases:*

- (i) *If $0 = k < l \leq n$ or $0 \leq k < l = n$ and $l - k \geq 2$.*
- (ii) *If $1 \leq k < l \leq n - 1$ and $l - k \geq 2$ is even.*

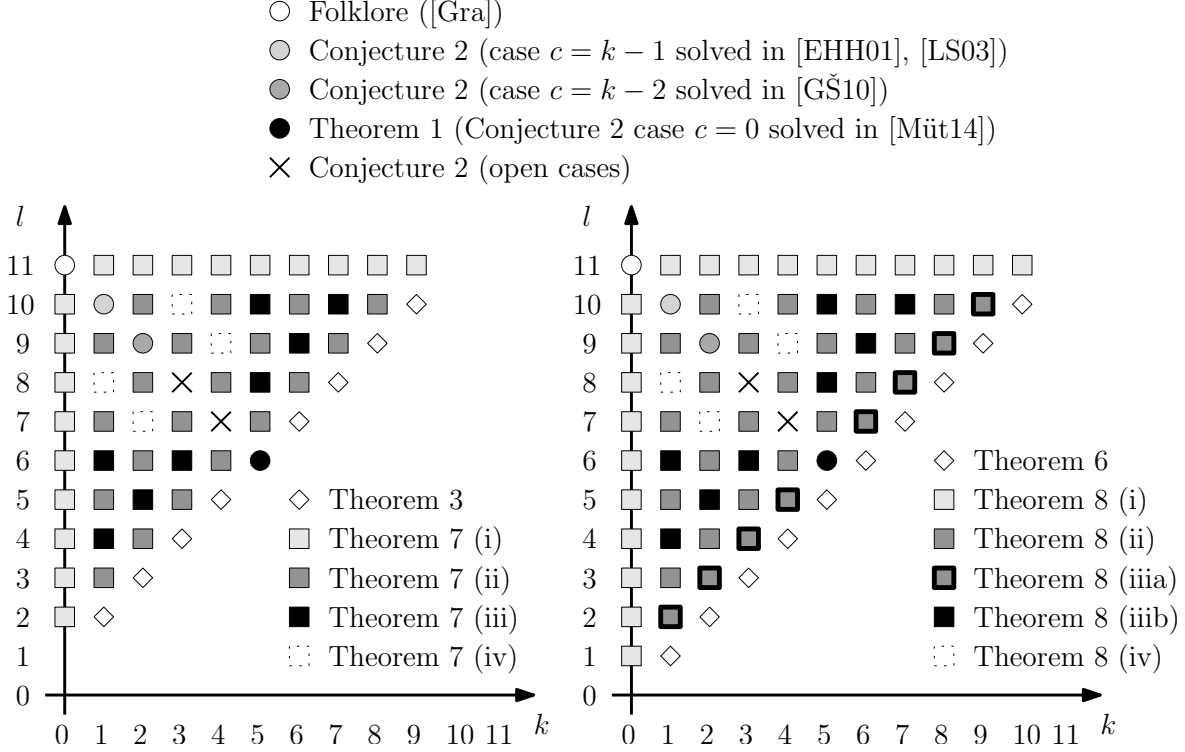


FIGURE 1. The different cases of k and l covered by our two main theorems on saturating cycles (left, Theorem 7) and on tight enumerations (right, Theorem 8) in $Q_{n,[k,l]}$ for the case $n = 11$. A more extensive animation of the entire parameter space (n, k, l) is available on the second author's website [www].

- (iii) If $1 \leq k < l \leq \lceil n/2 \rceil$ or $\lfloor n/2 \rfloor \leq k < l \leq n - 1$ and $l - k \geq 3$ is odd.
 (iv) If $1 \leq k < l \leq n - 1$ and $l - k \geq 3$ is odd, under the additional assumption that $Q_{2m+1,[m-c,m+1+c]}$, $c := (l - k - 1)/2$, has a Hamilton cycle for all $m = c, c + 1, \dots, (\min(k + l, 2n - k - l) - 1)/2$.

Our second set of results resolves Question 5 positively for all possible values of k and l except the cases covered by Conjecture 2 (the case $l = k$ is already covered by Theorem 6), see the right-hand side of Figure 1.

Theorem 8. For any $n \geq 3$ there is a tight enumeration of the vertices of $Q_{n,[k,l]}$ in the following cases:

- (i) If $0 = k < l \leq n$ or $0 \leq k < l = n$.
 (ii) If $1 \leq k < l \leq n$ and $l - k \geq 2$ is even.
 (iii) If $1 \leq k < l \leq n - 1$ and $l - k = 1$.
 (iiib) If $1 \leq k < l \leq \lceil n/2 \rceil$ or $\lfloor n/2 \rfloor \leq k < l \leq n - 1$ and $l - k \geq 3$ is odd.
 (iv) If $1 \leq k < l \leq n - 1$ and $l - k \geq 3$ is odd, under the additional assumption that $Q_{2m+1,[m-c,m+1+c]}$, $c := (l - k - 1)/2$, has a Hamilton cycle for all $m = c, c + 1, \dots, (\min(k + l, 2n - k - l) - 1)/2$.

Note that the last part (iv) of Theorems 7 and 8 is conditional on the validity of Conjecture 2. In fact, by what we said before (recall the paragraph below Conjecture 2) we know that the additional assumption in (iv) is satisfied for $m \in \{c, c + 1, c + 2\}$ (so the statement could be slightly strengthened).

The tight enumerations we construct to prove Theorem 8 have the additional property that all distance-2 steps are within single levels (but never between two different levels k and $k + 2$). In terms of sets, these steps therefore correspond to exchanging a single element.

For all the unconditional results in Theorems 7 and 8 we provide corresponding optimal generation algorithms.

Theorem 9. (a) For any interval $[k, l]$ as in case (i) or (ii) of Theorems 7 and 8, respectively, there is a corresponding loopless algorithm that generates each bitstring of a saturating cycle or a tight enumeration of the vertices of $Q_{n,[k,l]}$ in time $\mathcal{O}(1)$.

(b) For any interval $[k, l]$ as in case (iii) of Theorems 7 and 8, respectively, there is a corresponding algorithm that generates each bitstring of a saturating cycle or a tight enumeration of the vertices of $Q_{n,[k,l]}$ in time $\mathcal{O}(1)$ on average.

It should be noted that the algorithms for part (a) of Theorem 9 are considerably simpler than those for part (b). The reason is that the underlying constructions are entirely different. In particular, for part (b) we repeatedly call the (average) constant-time algorithm to compute a Hamilton cycle in $Q_{2m+1,[m,m+1]}$, $m \leq \lfloor (n-1)/2 \rfloor$, an algorithmic version of Theorem 1, presented in [MN15, MN16] (and this algorithm is admittedly rather complex). The initialization time of our algorithms is $\mathcal{O}(n)$, and the required space is $\mathcal{O}(n)$.

We implemented all these algorithms in C++, and we invite the reader to experiment with this code, which can be found on our website [www].

In view of these results, the only remaining (and therefore even more interesting) open case is the question whether the cube of odd dimension has a Hamilton cycle between any two symmetric levels, i.e., Conjecture 2 (these open cases are represented by crosses in Figure 1). Given the results from [GS10] and [Müt16], the next natural step towards resolving this conjecture would be to investigate whether the graphs $Q_{2k+1,[3,2k-2]}$ or $Q_{2k+1,[k-1,k+2]}$ have a Hamilton cycle for all $k \geq 1$.

In this paper we provide the following partial result towards the general conjecture: We show the existence of long cycles in the graph $Q_{2k+1,[k-c,k+1+c]}$, $c \in \{0, 1, \dots, k\}$. This approximate version of the conjecture is similar in spirit to the line of work [Sav93, FT95, SW95, Joh04] that preceded the proof of Theorem 1.

Theorem 10. For any $k \geq 1$ and $c \in \{0, 1, \dots, k\}$, the graph $Q_{2k+1,[k-c,k+1+c]}$ has a cycle that visits at least a $(1 - \epsilon)$ -fraction of all vertices, where $\epsilon := \frac{1}{2(c+1)} \min(1, \exp(\frac{(c+1)^2}{k-c}) - 1)$. In particular, for any c and $k \rightarrow \infty$, the cycle visits a $(1 - o(1))$ -fraction of all vertices.

1.2. Related work. In [Ehr73] an algorithm is presented that generates the vertices of $Q_{n,[k,l]}$ (for an arbitrary interval $[k, l]$) such that any two consecutive vertices have Hamming distance 1 or 2, where the value 2 appears only between vertices on level k and l , but the Hamming distance between the first and last vertex is arbitrary (possibly n). The running time of this algorithm is $\mathcal{O}(n)$ per generated vertex. In addition, this paper presents a loopless algorithm (time $\mathcal{O}(1)$ per generated vertex) to generate all vertices in $Q_{n,[k,l]}$ level by level, using only distance-2 steps in each level. In particular, these enumerations are not cycles in $Q_{n,[k,l]}$, and they are not tight.

In [SW14] the authors present algorithms for enumerating all vertices of $Q_{n,[k,l]}$ (for an arbitrary interval $[k, l]$) such that any two consecutive bitstrings have Levenshtein distance at most 2 and Hamming distance at most 4. The Levenshtein distance is the minimum number of bit insertions, deletions, and bitflips necessary to transform one bitstring into the other. Again, these enumerations are not cycles in $Q_{n,[k,l]}$ and they are not tight. However, the corresponding generation algorithms are very simple and fast (loopless). Improving on this, as a byproduct of the results mentioned in the previous section we obtain a simple loopless algorithm to enumerate all vertices of $Q_{n,[k,l]}$ (for an arbitrary interval $[k, l]$) such that any two consecutive bitstrings have Hamming distance (and Levenshtein distance) at most 2.

1.3. Outline of this paper. In Sections 2 and 3 we present the proofs of Theorems 7 and 8, respectively. In Section 4 we provide the corresponding optimal generation algorithms, proving Theorem 9. Finally, in Section 5 we prove Theorem 10.

2. SATURATING CYCLES

2.1. Trimming Gray codes and proofs of Theorem 7 (i)+(ii). In this section we prove cases (i) and (ii) from Theorem 7 by showing that the standard reflected Gray code in Q_n mentioned in the introduction (see [Gra] and [Knu11, Section 7.2.1.1]) can be ‘trimmed’ to any number of consecutive levels of Q_n so that it visits all these vertices except possibly some vertices from the first and last levels. This technique is a generalization of the approach presented in [TL73] to prove Theorem 6, and it yields the following result:

Theorem 11. *For any $n \geq 3$ and k, l with $0 \leq k < l \leq n$ and $l - k \geq 2$, the graph $Q_{n,[k,l]}$ has a cycle that visits all vertices except possibly some vertices from levels k and l .*

Note that if $l - k$ is even, then the first and last level of $Q_{n,[k,l]}$ are from the same bipartite class, so the cycle obtained from Theorem 11 is saturating, which immediately yields Theorem 7 (ii).

We begin by introducing a few definitions. For any graph G whose vertices are bitstrings and for any bitstring x , we write $G \circ x$ for the graph obtained from G by concatenating every vertex with x . For $b \in \{0, 1\}$ and any integer $k \geq 0$ we let b^k denote the bitstring that consists of exactly k many b -bits. For any two bitstrings x and y , we let $d(x, y)$ denote the number of positions in which x and y differ (their Hamming distance).

For any sequence Γ of (not necessarily distinct) vertices in a graph we let $f(\Gamma)$ and $\ell(\Gamma)$ denote the first and the last entry of Γ . We also define $\text{rev}(\Gamma)$ as the reversed sequence of vertices, i.e., $f(\text{rev}(\Gamma)) = \ell(\Gamma)$ and $\ell(\text{rev}(\Gamma)) = f(\Gamma)$. Moreover, we let $s_\Gamma(x)$ and $p_\Gamma(x)$, respectively, denote the successor and predecessor of the vertex x in Γ , where we define cyclically $s_\Gamma(\ell(\Gamma)) = f(\Gamma)$ and $p_\Gamma(f(\Gamma)) = \ell(\Gamma)$. We may omit the subscript Γ whenever it is clear from the context. For two sequences Γ and Γ' we let (Γ, Γ') denote their concatenation. Furthermore, if Γ is nonempty then we let Γ^- denote the sequence obtained from Γ by removing the last element, i.e., $\Gamma = (\Gamma^-, \ell(\Gamma))$.

The (n -dimensional) *reflected Gray code* Γ_n is a (cyclic) sequence Γ_n of all vertices of Q_n defined recursively by

$$\Gamma_1 = (0, 1) , \tag{1a}$$

$$\Gamma_{n+1} = (\Gamma_n \circ 0, \text{rev}(\Gamma_n) \circ 1) , \quad n \geq 1 . \tag{1b}$$

In words, Γ_{n+1} is the concatenation of Γ_n where each vertex is augmented with an additional 0-bit in the last coordinate and the reverse of Γ_n augmented with an additional 1-bit. See Figure 2 for an illustration.

The reflected Gray code Γ_n is the standard example how to enumerate all bitstrings of length n such that any two consecutive bitstrings differ in exactly one bit. For an explicit definition of Γ_n and further interesting properties see [Knu11, Section 7.2.1.1].

For any $0 \leq k \leq n$ let $\Gamma_{n,k}$ be the subsequence of Γ_n that contains all vertices in level k . From (1) it follows that

$$\Gamma_{1,0} = (0) , \quad \Gamma_{1,1} = (1) , \tag{2a}$$

$$\Gamma_{n+1,k} = (\Gamma_{n,k} \circ 0, \text{rev}(\Gamma_{n,k-1}) \circ 1) , \quad n \geq 1 \text{ and } 0 \leq k \leq n+1 , \tag{2b}$$

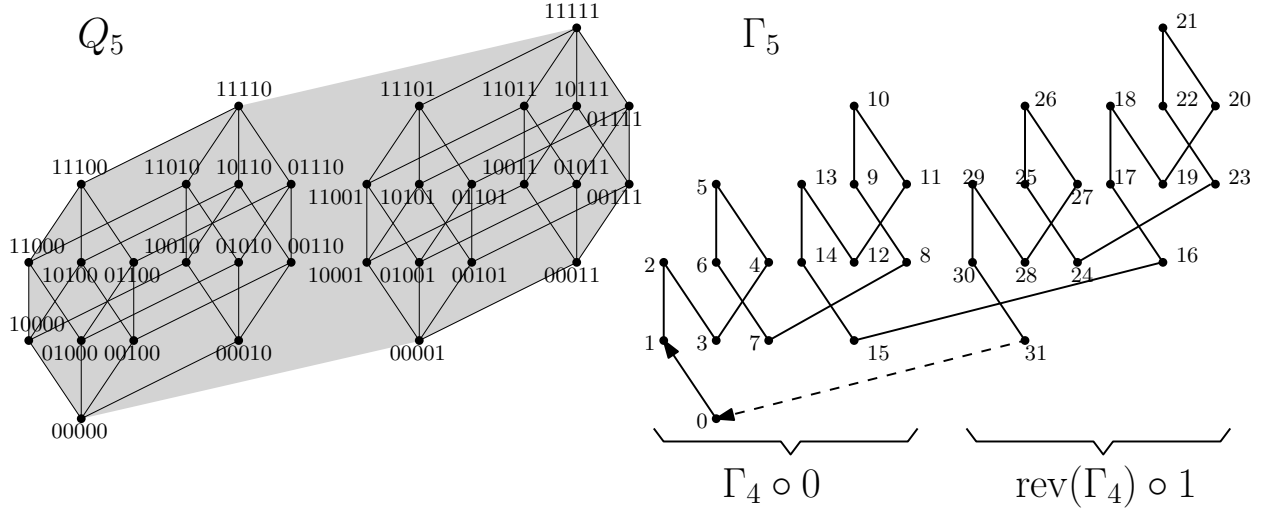


FIGURE 2. The hypercube Q_5 (left), where the grey area represents all 16 edges along which the last bit is flipped, and the reflected Gray code Γ_5 in Q_5 (right), where the numbers are indices of vertices in Γ_5 (starting from 0). The dashed edge represents the adjacency between the last and first vertex of Γ_5 .

where for unified treatment of border cases we define $\Gamma_{n,k} := ()$ (i.e., the empty sequence) whenever $k < 0$ or $k > n$. Furthermore, by (2) we have

$$f(\Gamma_{n,k}) = 1^k 0^{n-k} \quad , \quad 0 \leq k \leq n \quad , \quad (3a)$$

$$\ell(\Gamma_{n,k}) = 1^{k-1} 0^{n-k} 1 \quad , \quad 0 < k \leq n \quad . \quad (3b)$$

$$\ell(\Gamma_{n,0}) = f(\Gamma_{n,0}) = 0^n \quad . \quad (3c)$$

See Figure 3 for an illustration.

As already observed in [TL73], any two consecutive vertices in $\Gamma_{n,k}$ differ in exactly two positions. The sequence $\Gamma_{n,k}$ therefore provides an enumeration of all k -element subsets of $[n]$ such that any two consecutive k -sets differ in exchanging a single element (recall Theorem 6). For the purpose of self-containment we rephrase the inductive argument from [TL73].

Lemma 12 ([TL73]). *For any $n \geq 2$ and $0 < k < n$ let x be a vertex of $\Gamma_{n,k}$ and $y := s_{\Gamma_{n,k}}(x)$. Then we have $d(x, y) = 2$.*

Proof. We prove the lemma by induction on n . The statement trivially holds for $n = 2$, as $\Gamma_{2,1} = (10, 01)$, settling the induction basis. For the induction step we assume that the statement holds for some $n \geq 2$ and all $0 < k < n$, and we show that it also holds for $n + 1$ and all $0 < k < n + 1$. By (2b) it suffices to consider only the vertices $x = \ell(\Gamma_{n,k} \circ 0)$ and $x = \ell(\text{rev}(\Gamma_{n,k-1}) \circ 1)$. For all other choices of x the claim follows easily by induction. For the case $x = \ell(\Gamma_{n,k} \circ 0)$ we obtain

$$x = \ell(\Gamma_{n,k}) \circ 0 \stackrel{(3b)}{=} 1^{k-1} 0^{n-k} 10 \quad , \quad (4)$$

for all $0 < k < n + 1$ and

$$y = s_{\Gamma_{n,k}}(x) \stackrel{(2b)}{=} f(\text{rev}(\Gamma_{n,k-1})) \circ 1 = \ell(\Gamma_{n,k-1}) \circ 1 \stackrel{(3b),(3c)}{=} \begin{cases} 1^{k-2} 0^{n-k+1} 11 & \text{if } 1 < k < n + 1 \quad , \\ 0^n 1 & \text{if } k = 1 \quad . \end{cases} \quad (5)$$

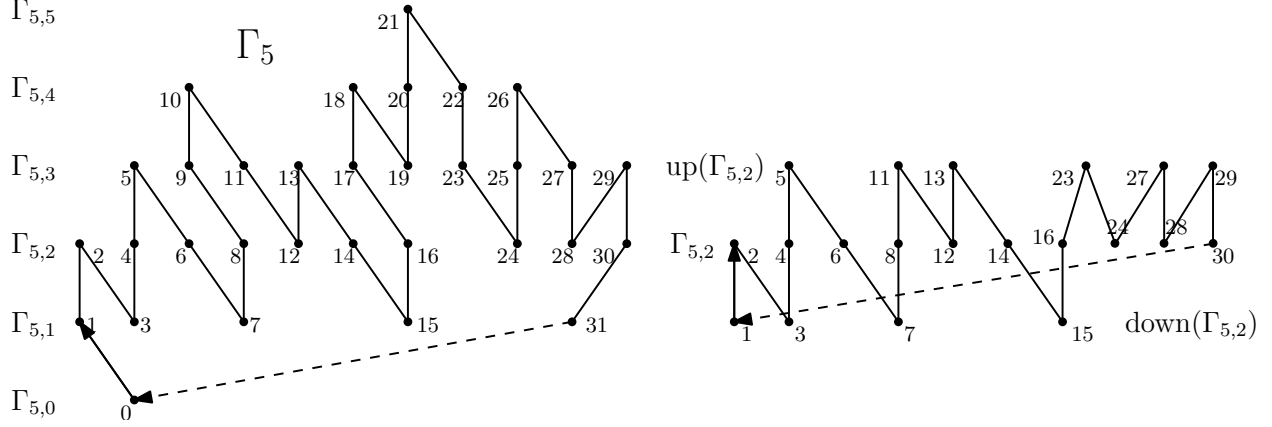


FIGURE 3. Schematic drawing of Γ_5 (left) highlighting the order in which levels are visited, and the corresponding sequences $\Gamma_{5,k}$, $0 \leq k \leq 5$. See Figure 2 what the actual vertices are. The sequences $\text{up}(\Gamma_{5,2})$, $\text{down}(\Gamma_{5,2})$, and Γ_5 trimmed to levels 1 up to 3 of Q_5 (right).

Comparing the right-hand sides of (4) and (5) shows that indeed $d(x, y) = 2$. For the case $x = \ell(\text{rev}(\Gamma_{n,k-1}) \circ 1)$ we obtain

$$x = \ell(\text{rev}(\Gamma_{n,k-1})) \circ 1 = f(\Gamma_{n,k-1}) \circ 1 \stackrel{(3a)}{=} 1^{k-1} 0^{n-k+1} 1 \quad (6)$$

and

$$y = s_{\Gamma_{n,k}}(x) \stackrel{(2b)}{=} f(\Gamma_{n,k}) \circ 0 \stackrel{(3a)}{=} 1^k 0^{n-k} 0 \quad (7)$$

for all $0 < k < n + 1$. Comparing the right-hand sides of (6) and (7) also yields $d(x, y) = 2$, as desired. This completes the proof of the lemma. \square

Clearly, any two vertices x, y in level k at distance 2 have a unique common neighbor in level $k - 1$ and a unique common neighbor in level $k + 1$, let us denote them by $\text{down}(x, y)$ and $\text{up}(x, y)$, respectively. The key idea in trimming the reflected Gray code to a given sequence of consecutive levels $[k, l]$, where $l - k \geq 2$, is to replace the subpath P of Γ_n in Q_n between a vertex x in level $l - 1$ and its consecutive vertex $s_{\Gamma_{n,l-1}}(x)$ by the path $(x, \text{up}(x, s(x)), s(x))$ if P ascends above level $l - 1$, and between a vertex x in level $k + 1$ and its consecutive vertex $s_{\Gamma_{n,k+1}}(x)$ by the path $(x, \text{down}(x, s(x)), s(x))$ if P descends below level $k + 1$. See Figure 3 for an illustration.

Formally, we say that a vertex x of Q_n in level k is an *upward vertex* if $s_{\Gamma_n}(x)$ is in level $k + 1$, and a *downward vertex* if $s_{\Gamma_n}(x)$ is in level $k - 1$ (no other case is possible). Thus, the reflected Gray code Γ_n ascends from upward vertices and descends from downward vertices. Note that $\ell(\Gamma_{n,k})$ is a downward vertex for every $0 < k \leq n$ since Γ_n starts in level 0 and ends in level 1. For any $0 < k \leq n$, we let $\text{up}(\Gamma_{n,k})$ denote the sequence of all vertices $\text{up}(x, s(x))$ in level $k + 1$, where x is an upward vertex of $\Gamma_{n,k}$, in the order induced by $\Gamma_{n,k}$. Similarly, for any $0 \leq k < n$ we let $\text{down}(\Gamma_{n,k})$ denote the sequence of all vertices $\text{down}(x, s(x))$ in level $k - 1$, where x is a downward vertex of $\Gamma_{n,k}$, in the order induced by $\Gamma_{n,k}$. Note that $\text{up}(\Gamma_{n,n-1}) = (1^n)$ and $\text{down}(\Gamma_{n,1}) = (0^n)$ for every $n \geq 2$. Moreover, we trivially have $\text{up}(\Gamma_{n,n}) = ()$ and $\text{down}(\Gamma_{n,0}) = ()$. Furthermore, observe that $\text{up}(\Gamma_{n,k})$ commutes with taking reverse and $\text{down}(\Gamma_{n,k})$ commutes with taking reverse up to the last vertex. More precisely, we have

$$\text{up}(\text{rev}(\Gamma_{n,k})) = \text{rev}(\text{up}(\Gamma_{n,k})) \quad , \quad 0 < k \leq n \quad , \quad (8a)$$

$$(\text{down}(\text{rev}(\Gamma_{n,k})))^- = \text{rev}((\text{down}(\Gamma_{n,k}))^-) \quad , \quad 0 \leq k < n \quad . \quad (8b)$$

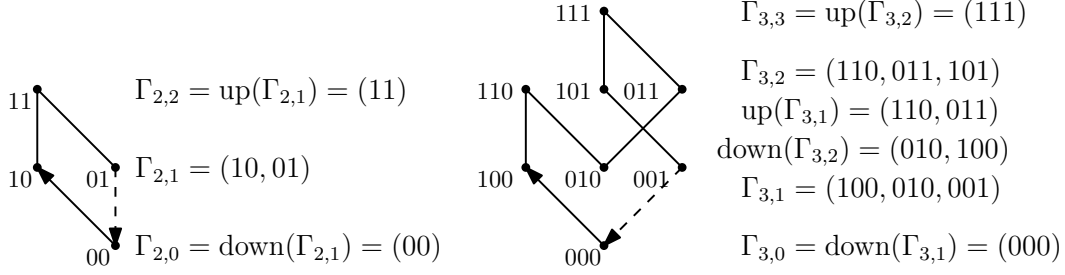


FIGURE 4. Γ_n , $\Gamma_{n,k}$, $\text{up}(\Gamma_{n,k})$, and $\text{down}(\Gamma_{n,k})$ for $n = 2$ (left) and $n = 3$ (right) and all relevant values of k .

Moreover, we know that

$$\ell(\text{down}(\Gamma_{n,k})) = \text{down}(f(\Gamma_{n,k}), \ell(\Gamma_{n,k})) = \text{down}(\ell(\text{rev}(\Gamma_{n,k})), f(\text{rev}(\Gamma_{n,k}))) = \ell(\text{down}(\text{rev}(\Gamma_{n,k}))) . \quad (9)$$

The next lemma captures the key property guaranteeing that in trimming the reflected Gray code as described above we will never visit the same vertex twice.

Lemma 13. *For any $n \geq 2$ and $0 < k < n$, $\text{up}(\Gamma_{n,k})$ is a subsequence of $\Gamma_{n,k+1}$. Moreover, $(\ell(\text{down}(\Gamma_{n,k})), (\text{down}(\Gamma_{n,k}))^-)$ is a subsequence of $\Gamma_{n,k-1}$, we have $\ell(\text{down}(\Gamma_{n,k})) = f(\Gamma_{n,k-1})$, and the sequence $\text{down}(\Gamma_{n,k})$ does not contain the vertex $\ell(\Gamma_{n,k-1})$ if $k > 1$.*

Note that the sequence $(\ell(\text{down}(\Gamma_{n,k})), (\text{down}(\Gamma_{n,k}))^-)$ referred to in Lemma 13 is simply $\text{down}(\Gamma_{n,k})$ rotated to the right once.

Proof. We proceed by induction on n . Figure 4 shows that the statement holds for $n = 2$ and $n = 3$, settling the induction basis. We now assume that statement holds for some $n \geq 3$ and all $0 < k < n$, and show that it also holds for $n + 1$ and all $0 < k < n + 1$.

For $1 < k < n + 1$, since both $\ell(\Gamma_{n,k})$ and $\ell(\Gamma_{n+1,k})$ are downward vertices, we have

$$\text{up}(\Gamma_{n+1,k}) \stackrel{(2b)}{=} (\text{up}(\Gamma_{n,k} \circ 0), \text{up}(\text{rev}(\Gamma_{n,k-1}) \circ 1)) \stackrel{(8a)}{=} (\text{up}(\Gamma_{n,k} \circ 0), \text{rev}(\text{up}(\Gamma_{n,k-1} \circ 1))) . \quad (10)$$

By induction, $\text{up}(\Gamma_{n,k})$ and $\text{up}(\Gamma_{n,k-1})$ are subsequences of $\Gamma_{n,k+1}$ and $\Gamma_{n,k}$, respectively (for $k = n$ the sequence $\text{up}(\Gamma_{n,k})$ is empty), so we obtain from (2b) and (10) that $\text{up}(\Gamma_{n+1,k})$ is a subsequence of $\Gamma_{n+1,k+1}$. For $k = 1$ the vertex $\ell(\Gamma_{n,1})$ is a downward vertex in Γ_n , but not in Γ_{n+1} . In fact, in $\Gamma_{n+1,1}$ there is only one more vertex after $\ell(\Gamma_{n,1}) \circ 0 = 0^{n-1}10$, namely the vertex $f(\Gamma_n) \circ 1 = 0^n1$ (recall (2b) and (3)). In this case we therefore have

$$\text{up}(\Gamma_{n+1,1}) \stackrel{(2b)}{=} (\text{up}(\Gamma_{n,1} \circ 0), \text{up}(0^{n-1}10, 0^n1)) = (\text{up}(\Gamma_{n,1} \circ 0), 0^{n-1}11) , \quad (11)$$

and since $\Gamma_{n+1,2}$ visits all vertices ending with a 0-bit before all vertices ending with a 1-bit, we conclude from (11) that $\text{up}(\Gamma_{n+1,1})$ is indeed a subsequence of $\Gamma_{n+1,2}$. This completes the proof of the first part of the lemma.

For $k = 1$ we have $\text{down}(\Gamma_{n+1,1}) = (0^{n+1})$, which is indeed a subsequence of $\Gamma_{n+1,0} = (0^{n+1})$. We now assume that $1 < k < n + 1$. Here, the argument for downward vertices is more complicated than for upward vertices since the successors of $\ell(\Gamma_{n,k} \circ 0)$, $\ell(\text{rev}(\Gamma_{n,k-1} \circ 1))$ change with the concatenation of $\Gamma_{n,k} \circ 0$ and $\text{rev}(\Gamma_{n,k-1} \circ 1)$. By the induction hypothesis, (8b) and (9) we have

$$\text{down}(\Gamma_{n,k} \circ 0) = ((\text{down}(\Gamma_{n,k} \circ 0))^- , f(\Gamma_{n,k-1} \circ 0)) , \quad (12a)$$

$$\text{down}(\text{rev}(\Gamma_{n,k-1} \circ 1)) = (\text{rev}((\text{down}(\Gamma_{n,k-1} \circ 1))^-) , f(\Gamma_{n,k-2} \circ 1)) . \quad (12b)$$

From (12a) we obtain that the vertex $f(\Gamma_{n,k-1} \circ 0) = f(\Gamma_{n+1,k-1})$ is not contained in the sequence $(\text{down}(\Gamma_{n,k} \circ 0))^-$ and from (12b) that the vertex $f(\Gamma_{n,k-2} \circ 1) = \ell(\Gamma_{n+1,k-1})$ is not contained in the sequence $\text{rev}((\text{down}(\Gamma_{n,k-1} \circ 1))^-)$ (recall (3a) and (3b)).

We now compute the two vertices in $\text{down}(\Gamma_{n+1,k})$ added between the boundaries of $\text{down}(\Gamma_{n,k} \circ 0)$ and $\text{down}(\text{rev}(\Gamma_{n,k-1} \circ 1))$ in the induction step (2b). These two vertices are

$$\begin{aligned} \text{down}(\ell(\Gamma_{n,k} \circ 0), f(\text{rev}(\Gamma_{n,k-1} \circ 1))) &= \text{down}(\ell(\Gamma_{n,k} \circ 0), \ell(\Gamma_{n,k-1} \circ 1)) \\ &\stackrel{(3b)}{=} \text{down}(1^{k-1}0^{n-k}10, 1^{k-2}0^{n-k+1}11) \\ &= 1^{k-2}0^{n-k+1}10 \stackrel{(3b)}{=} \ell(\Gamma_{n,k-1} \circ 0) , \end{aligned} \quad (13)$$

$$\begin{aligned} \text{down}(\ell(\text{rev}(\Gamma_{n,k-1} \circ 1)), f(\Gamma_{n,k} \circ 0)) &= \text{down}(f(\Gamma_{n,k-1} \circ 1), f(\Gamma_{n,k} \circ 0)) \\ &\stackrel{(3a)}{=} \text{down}(1^{k-1}0^{n-k+1}1, 1^k0^{n-k}0) \\ &= 1^{k-1}0^{n-k+2} \stackrel{(3a)}{=} f(\Gamma_{n,k-1} \circ 0) . \end{aligned} \quad (14)$$

Combining our previous observations, we obtain

$$\begin{aligned} \text{down}(\Gamma_{n+1,k}) &\stackrel{(2b)}{=} \text{down}((\Gamma_{n,k} \circ 0, \text{rev}(\Gamma_{n,k-1} \circ 1))) \\ &= \left((\text{down}(\Gamma_{n,k} \circ 0))^- , \text{down}(\ell(\Gamma_{n,k} \circ 0), f(\text{rev}(\Gamma_{n,k-1} \circ 1))), (\text{down}(\text{rev}(\Gamma_{n,k-1} \circ 1)))^- , \right. \\ &\quad \left. \text{down}(\ell(\text{rev}(\Gamma_{n,k-1} \circ 1)), f(\Gamma_{n,k} \circ 0)) \right) \\ &\stackrel{(8b),(13),(14)}{=} ((\text{down}(\Gamma_{n,k} \circ 0))^- , \ell(\Gamma_{n,k-1} \circ 0), \text{rev}((\text{down}(\Gamma_{n,k-1} \circ 1))^-), f(\Gamma_{n,k-1} \circ 0)) . \end{aligned} \quad (15)$$

As we observed before, the vertex $f(\Gamma_{n,k-1} \circ 0)$ is not contained in the sequence $(\text{down}(\Gamma_{n,k} \circ 0))^-$, and the vertex $\ell(\Gamma_{n,k-1} \circ 0)$ is not contained in this sequence by induction, implying that (15) is a sequence of distinct vertices in level $k-1$ of Q_{n+1} . Moreover, as $(\text{down}(\Gamma_{n,k} \circ 0))^-$ and $(\text{down}(\Gamma_{n,k-1} \circ 1))^-$ are subsequences of $\Gamma_{n,k-1} \circ 0$ and $\Gamma_{n,k-2} \circ 1$, respectively, by induction, we obtain from (2b) and (15) that $(\ell(\text{down}(\Gamma_{n+1,k})), (\text{down}(\Gamma_{n+1,k}))^-)$ is a subsequence of $\Gamma_{n+1,k-1}$, as claimed. Furthermore, also from (15) we know that $\ell(\text{down}(\Gamma_{n+1,k})) = f(\Gamma_{n,k-1} \circ 0) = f(\Gamma_{n+1,k-1})$ (recall (3a)), as desired. Lastly, as observed before, the vertex $\ell(\Gamma_{n+1,k-1})$ is not contained in the sequence $\text{rev}((\text{down}(\Gamma_{n,k-1} \circ 1))^-)$ and its last bit is 1, so it is not contained in the sequence $\text{down}(\Gamma_{n+1,k})$ either.

This completes the proof of the lemma. \square

We now strengthen the first part of the statement of Lemma 13 for upward vertices even further, which will be needed later.

Lemma 14. *For any $n \geq 2$ and $0 < k < n$ let x be an upward vertex of $\Gamma_{n,k}$ and $y := s_{\Gamma_{n,k}}(x)$. Then $\text{up}(x, y)$ equals $s_{\Gamma_n}(x)$ or $p_{\Gamma_n}(y)$.*

In words, Lemma 14 asserts that the vertex $\text{up}(x, y)$ equals the successor of x or the predecessor of y on Γ_n . Note that this implies in particular that $\text{up}(\Gamma_{n,k})$ is a subsequence of $\Gamma_{n,k+1}$. We shall see later that the statement of Lemma 14 can be strengthened even further (see (22) below); specifically, whether $\text{up}(x, y)$ equals $s_{\Gamma_n}(x)$ or $p_{\Gamma_n}(y)$ is determined only by the parity of k .

Proof. Clearly, for any sequence Γ of vertices and any vertex x in Γ we have

$$s_{\Gamma}(x) = p_{\text{rev}(\Gamma)}(x) , \quad p_{\Gamma}(x) = s_{\text{rev}(\Gamma)}(x) . \quad (16)$$

We prove the lemma by induction on n . Figure 4 shows that the statement holds for $n = 2$ and $n = 3$, settling the induction basis. For the induction step we assume that the statement holds for some $n \geq 3$ and all $0 < k < n$, and we show that it also holds for $n + 1$ and all $0 < k < n + 1$.

First we consider an upward vertex x in $\Gamma_{n+1,k}$ for $1 < k < n + 1$. Recall that $\ell(\Gamma_{n,k})$ is a downward vertex, so by (2b) x must be different from $\ell(\Gamma_{n,k}) \circ 0$. Also, x must be different from $\ell(\Gamma_{n+1,k})$ (which by (2b) equals $f(\Gamma_{n,k-1}) \circ 1$). Using (16) the claim therefore follows by induction from (2b). Now consider an upward vertex x in $\Gamma_{n+1,1}$. The only case here where the previous argument fails is that x might be equal to $\ell(\Gamma_{n,1}) \circ 0 = 0^{n-1}10$ (recall (3b)). This is because even though $\ell(\Gamma_{n,1}) = 0^{n-1}1$ is a downward vertex in $\Gamma_{n,1}$, since it is the last vertex of Γ_n it will become an upward vertex in $\Gamma_{n+1,1}$, i.e., in Γ_{n+1} the vertex $x = 0^{n-1}10$ will be followed by $z := 0^{n-1}11$, and there is only one more entry of $\Gamma_{n+1,1}$ after x , namely $y = f(\Gamma_{n,0}) \circ 1 = 0^n1$ (recall (2b) and (3a)). And indeed, the vertices x , y and z satisfy the relation $s_{\Gamma_{n+1}}(x) = z = \text{up}(x, y)$. \square

We are now ready to prove Theorem 11.

Proof of Theorem 11. We build the desired cycle by trimming the reflected Gray code Γ_n to levels $[k, l]$. Subpaths of Γ_n within the levels $[k + 1, l - 1]$ remain unchanged (including the orientation). Each subpath P of Γ_n that descends from some downward vertex x at level $k + 1$ to lower levels returns back to level $k + 1$ at the vertex $y := s_{\Gamma_{n,k+1}}(x)$. Since $d(x, y) = 2$ by Lemma 12, we may replace P by the path $P' = (x, \text{down}(x, y), y)$. Note that P' has the same end vertices and orientation as P , and visits only a single vertex at level k . After trimming all these descending paths, we visit on level k precisely the vertices of $\text{down}(\Gamma_{n,k+1})$. Since $\text{down}(\Gamma_{n,k+1})$ (after one rotation to the right) is a subsequence of $\Gamma_{n,k}$ by Lemma 13, all these vertices are distinct, and hence distinct trimmed paths may have at most end vertices in level k in common.

Similar arguments apply for trimming subpaths of Γ_n ascending from upward vertices at level $l - 1$ to levels above. In this case the trimmed subpaths visit at level l precisely the vertices of $\text{up}(\Gamma_{n,l-1})$, and since this is a subsequence of $\Gamma_{n,l}$ by Lemma 13, all these vertices are distinct. Therefore trimming correctly produces a cycle visiting all vertices in levels $[k, l]$ except the vertices from level k that are not in $\text{down}(\Gamma_{n,k+1})$ and the vertices from level l that are not in $\text{up}(\Gamma_{n,l-1})$. \square

Proof of Theorem 7 (ii). Follows immediately from Theorem 11, using that if $l - k$ is even, then the first and last level of $Q_{n,[k,l]}$ are from the same bipartite class. \square

Proof of Theorem 7 (i). We only consider the case $0 = k < l \leq n$, the other case follows by symmetry, using that $Q_{n,[k,l]}$ is isomorphic to $Q_{n,[n-l,n-k]}$. The proof proceeds very similarly to the proof of Theorem 11, but we only trim the subpaths of Γ_n ascending above level $l - 1$, so that the highest level where vertices are visited is level l (no trimming is applied at the bottom level 0). We therefore obtain a cycle that visits all vertices in levels $[0, l]$, except the vertices from level l that are not in $\text{up}(\Gamma_{n,l-1})$. As the cycle omits only vertices from level l , it must be saturating. \square

2.2. Gluing saturating cycles and proof of Theorem 7 (iii)+(iv). Trimming the reflected Gray code Γ_n to levels $[k, l]$ as described in the last section does not yield a saturating cycle when $l - k \geq 3$ is odd unless $k = 0$ or $l = n$. In general the trimmed cycle omits some vertices from both levels k and l , which are from different bipartite classes for odd $l - k$. We therefore use a different strategy to prove Theorem 7 (iii): We glue together several saturating cycles obtained from Theorem 3. However, this gluing approach yields a saturating cycle only if all involved levels are either below or above the middle (this is reflected by the conditions $l \leq \lfloor n/2 \rfloor$ or $\lfloor n/2 \rfloor \leq k$ in Theorem 7 (iii)). Otherwise the omitted vertices would be from different bipartite classes, so the resulting cycle would not be saturating. To prove Theorem 7 (iv), we inductively glue together

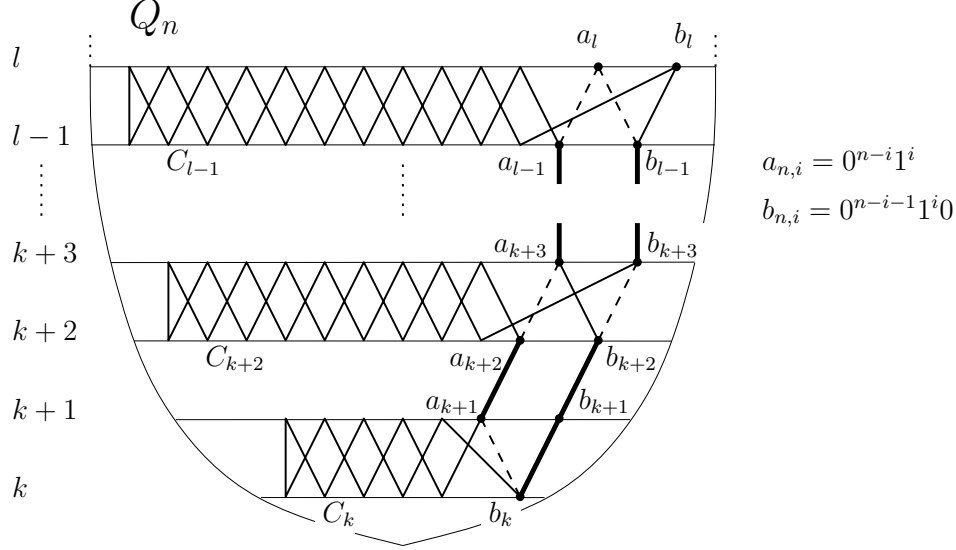


FIGURE 5. Notations used in the proof of Theorem 7 (iii). The removed edges are dashed, the added edges are bold.

pairs of saturating cycles of smaller dimension. Both proofs are similar to the approach presented in [MS15].

Recall that for any bipartite graph G , $v(G)$ denotes the number of vertices of G and $\delta(G)$ denotes the difference between the larger and the smaller partition class. We easily compute

$$v(Q_{n,[k,l]}) = \sum_{i=k}^l \binom{n}{i}, \quad (17a)$$

$$\delta(Q_{n,[k,l]}) = \left| \sum_{i=k}^l (-1)^i \binom{n}{i} \right| = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{l} & \text{if } l-k \text{ is even,} \\ \left| \binom{n-1}{k-1} - \binom{n-1}{l} \right| & \text{if } l-k \text{ is odd,} \end{cases} \quad (17b)$$

where the degenerate cases $k = 0$ and $l = n$ have to be treated by defining $\binom{n-1}{-1} = \binom{n-1}{n} = 0$. Clearly, $Q_{n,[k,l]}$ can have a Hamilton cycle only if $\delta(Q_{n,[k,l]}) = 0$, and from (17b) we conclude that this condition is satisfied if and only if n is odd and $l = n - k$ (Conjecture 2 says that this necessary condition is in fact sufficient for finding a Hamilton cycle).

Proof of Theorem 7 (iii). We only consider the case $1 \leq k < l \leq \lceil n/2 \rceil$, the other case follows by symmetry. By Theorem 3, there is a saturating cycle C_i between levels i and $i+1$ for every $i = k, k+2, \dots, l-1$. Note that each C_i visits all vertices in level i since $l \leq \lceil n/2 \rceil$. We now show how to join these $(l-k+1)/2 \geq 2$ many cycles to a single saturating cycle in $Q_{n,[k,l]}$. For the reader's convenience, the approach is illustrated in Figure 5.

For any level i of Q_n , we define two special vertices in this level:

$$a_{n,i} := 0^{n-i}1^i, \quad b_{n,i} := 0^{n-i-1}1^i0. \quad (18)$$

We omit the subscript n whenever it is clear from the context and simply write a_i or b_i in this case. In the lowest cycle C_k between levels k and $k+1$ we permute coordinates so that the vertices b_k and a_{k+1} are visited consecutively, and the vertex b_{k+1} is not visited at all. This is possible since permutation of coordinates is an automorphism of $Q_{n,[k,k+1]}$ and since there is a vertex in level $k+1$ that is not visited by C_k (and any neighbor of this vertex in level k is visited by C_k). Furthermore, in each of the other cycles C_i , $i = k+2, k+4, \dots, l-1$, we permute coordinates (independently

for each cycle) so that the vertices a_i, a_{i+1}, b_i and b_{i+1} are visited consecutively. This is possible since C_i contains a subpath of length 3 starting at level i and any path of length 3 between two consecutive levels has the property that the coordinate changes are in different directions along the three edges. We now modify the (permuted) cycles as follows: In the lowest cycle C_k we remove the edge (b_k, a_{k+1}) and replace it by the edge (b_k, b_{k+1}) (including the previously omitted vertex b_{k+1} in level $k+1$). In the intermediate cycles $C_i, i = k+2, k+4, \dots, l-3$, we remove the edges (a_i, a_{i+1}) and (b_i, b_{i+1}) . In the uppermost cycle C_{l-1} , we remove the edges (a_{l-1}, a_l) and (b_{l-1}, b_l) (creating a new omitted vertex a_l in level l). Finally, by adding the edges (a_{i-1}, a_i) and (b_{i-1}, b_i) for $i = k+2, k+4, \dots, l-1$ we join the resulting paths to a single cycle C . The cycle C is saturating, as the missed vertices all lie in levels $k+1, k+3, \dots, l$ by the condition $l \leq \lceil n/2 \rceil$; i.e., they all belong to the same bipartite class. \square

It remains to prove the last part (iv) of Theorem 7. Before doing so we introduce another definition. We say that a cycle C in $Q_{n,[k,l]}$ contains a *virtual 2-path* (u, v, w) , if (u, v, w) is a 2-path of $Q_{n,[k,l]}$ where u, w are in level l and v is in level $l-1$, the edge (u, v) is in C , but the vertex w is omitted by C .

Proof of Theorem 7 (iv). We only consider the case $k \leq n-l$. The case $k > n-l$ follows by symmetry. We inductively prove the following strengthening of the result: We additionally require that the only omitted vertices are in level l , and if there are such vertices that the cycle contains a virtual 2-path.

For any fixed (odd) value of $l-k \geq 3$ we prove this statement by induction on n . The induction basis for $n = l-k$, i.e. $k = 0$ and $l = n$, is settled by the reflected Gray code Γ_n .

Observe that for any $n \geq l-k$ and $k = 0$ this statement follows from Theorem 7 (i). As this cycle is obtained by trimming Γ_n to levels $[0, l]$ (recall the proof of Theorem 7 (i) from the previous section), the only omitted vertices are in level l , and it contains a virtual 2-path: Indeed, any omitted vertex in level l originates from replacing a path that leads from an upward vertex x in level $l-1$ to $s_{\Gamma_n}(x)$ in level l , and via some additional vertices in levels $[l, n]$ back to $p_{\Gamma_n}(y)$ in level l , where $y := s_{\Gamma_n, l-1}(x)$, and then to the vertex y in level $l-1$, by the shorter path $(x, \text{up}(x, y), y)$. By Lemma 14, this makes either $s_{\Gamma_n}(x)$ or $p_{\Gamma_n}(y)$ an omitted vertex in level l , and creates a virtual 2-path in both cases: If $\text{up}(x, y) = s_{\Gamma_n}(x)$, then $p_{\Gamma_n}(y)$ is omitted and $(s_{\Gamma_n}(x), y, p_{\Gamma_n}(y))$ is a virtual 2-path. Otherwise $\text{up}(x, y) = p_{\Gamma_n}(y)$ and $s_{\Gamma_n}(x)$ is omitted and $(p_{\Gamma_n}(y), x, s_{\Gamma_n}(x))$ is a virtual 2-path.

For any (odd) $n \geq l-k$ and $k = n-l$ the statement holds by the additional assumption (that there is a Hamilton cycle in $Q_{2m+1, [m-c, m+1+c]}$ for all $m = c, c+1, \dots, (n-1)/2$), and there are no omitted vertices at all.

We now assume that the statement holds for some $n \geq l-k$ and all relevant values of k , and show that it also holds for $n+1$ and $1 \leq k < (n+1)-l$. By induction we know that there is a saturating cycle C_0 in $Q_{n,[k,l]}$ (note that $k \leq n-l$; this cycle might actually be a Hamilton cycle). We also know that there is a saturating cycle C_1 in $Q_{n,[k-1, l-1]}$ (note that $k-1 \geq 0$) satisfying all the conditions of the theorem. In particular, there will be an omitted vertex x (recall (17b) and that $k-1 < n-(l-1)$) in level $l-1$ and a virtual 2-path. For any level i of Q_n , consider the special vertices $a_{n,i}$ and $b_{n,i}$ defined in (18). We permute coordinates in C_0 so that the vertices $a_{n, l-1}, a_{n, l}$ and $b_{n, l-1}$ are visited consecutively. Moreover, we permute coordinates in C_1 so that the virtual 2-path is mapped to $(a_{n, l-1}, b_{n, l-2}, b_{n, l-1})$. From the cycle C_0 we remove the edges $(a_{n, l-1}, a_{n, l})$ and $(b_{n, l-1}, a_{n, l})$ (creating a new omitted vertex $a_{n, l}$ in level l of Q_n), and we attach a 0-bit to all vertices. In the cycle C_1 we remove the edge $(b_{n, l-2}, a_{n, l-1})$ and replace it by the edge $(b_{n, l-2}, b_{n, l-1})$ (including the previously omitted vertex $b_{n, l-1}$ in level $l-1$ of Q_n), and we attach a 1-bit to all vertices. We connect the resulting paths by adding the edges $(a_{n, l-1} \circ 0, a_{n, l-1} \circ 1)$ and $(b_{n, l-1} \circ 0, b_{n, l-1} \circ 1)$, creating a cycle C in $Q_{n+1, [k, l]}$. The only omitted vertices of C are in level l of Q_{n+1} , so the cycle

is saturating. Moreover, $(a_{n,l-1} \circ 1, a_{n,l-1} \circ 0, a_{n,l} \circ 0)$ is a virtual 2-path in C . This completes the proof. \square

3. TIGHT ENUMERATIONS

We call a sequence C that contains each vertex of $Q_{n,[k,l]}$ exactly once an *enumeration of the vertices of $Q_{n,[k,l]}$* (recall that $s_C(\ell(C)) = f(C)$). The *total distance* of the enumeration C is $\text{td}(C) := \sum_{u \in C} d(u, s_C(u))$. As any two consecutive bitstrings in any enumeration have distance at least 1 and distance at least 2 if they are from the same bipartite class of $Q_{n,[k,l]}$, we have

$$\text{td}(C) \geq v(Q_{n,[k,l]}) + \delta(Q_{n,[k,l]}) \quad (19)$$

(recall (17)). A *tight enumeration* is an enumeration for which the lower bound (19) is attained. Clearly, an enumeration is tight, if and only if it has only distance-1 and distance-2 steps, and all the distance-2 steps are within the same partition class of $Q_{n,[k,l]}$ (this will be the larger partition class, and there will be exactly $\delta(Q_{n,[k,l]})$ such distance-2 steps in this class).

3.1. Proof of Theorem 8. For $k = l$ we have $v(Q_{n,[k,k]}) = \delta(Q_{n,[k,k]}) = \binom{n}{k}$ and a tight enumeration of all weight k bitstrings of length n is given by $\Gamma_{n,k}$, by Lemma 12 (this is exactly the proof of Theorem 6 presented in [TL73]).

We now proceed to prove cases (i)–(iv) of Theorem 8. The proofs are very similar to the corresponding proofs for cases (i)–(iv) of Theorem 7, and the key ideas are the same. There are however various technical subtleties to overcome.

Proof of Theorem 8 (ii). We trim the reflected Gray code Γ_n to levels $[k, l]$, but in a slightly different fashion from the proof of Theorem 11. Specifically, subpaths of Γ_n within the levels $[k, l]$ remain unchanged (including the orientation). Moreover, each subpath P of Γ_n that descends from a downward vertex x in level k returns back to level k at the vertex $y := s_{\Gamma_{n,k}}(x)$, and we replace P by the distance-2 step (x, y) (recall Lemma 12). Similarly, each subpath P of Γ_n that ascends from an upward vertex x in level l returns back to level l at the vertex $y := s_{\Gamma_{n,l}}(x)$, and we replace P by the distance-2 step (x, y) . This yields an enumeration C of all vertices of $Q_{n,[k,l]}$. Moreover, since $l - k$ is even, levels k and l of $Q_{n,[k,l]}$ belong to the same bipartite class, so all distance-2 steps of C are within the same bipartite class, and the remaining steps are distance-1 steps. It follows that C is a tight enumeration. \square

Proof of Theorem 8 (i). We only consider the case $0 = k < l \leq n$, the other case follows by symmetry. The proof proceeds very similarly as the proof of part (ii), but we only trim the subpaths of Γ_n ascending above level l (no trimming is applied at the bottom level 0). This yields an enumeration C of all vertices of $Q_{n,[0,l]}$. Moreover, all distance-2 steps of C are within the same bipartite class (in level l), and the remaining steps are distance-1 steps, implying that C is a tight enumeration. \square

Theorem 8 (iiia) is an immediate consequence of our next result, Theorem 15 below. This theorem is the analogue of Theorem 3 for tight enumerations. To state the result we need to introduce some notation: We say that an enumeration C of the vertices of $Q_{n,[k,k+1]}$ contains a *3-path* if C contains a contiguous subsequence (u, v, w, x) of vertices that form a path (of length 3) in $Q_{n,[k,k+1]}$ (in this order) where u, w are in level k , and v, x are in level $k + 1$. Furthermore, we say that C contains a *switched 2-path* if C contains a contiguous subsequence (u, v, w) of vertices such that (w, u, v) forms a path (of length 2) in $Q_{n,[k,k+1]}$ where u is in level k and v, w are in level $k + 1$.

Theorem 15. *For any $n \geq 3$ and $0 \leq k \leq n - 1$ there is a tight enumeration of the vertices in $Q_{n,[k,k+1]}$. Moreover, it contains a 3-path if $1 \leq k \leq n - 2$, and a switched 2-path if $k < \lfloor n/2 \rfloor$.*

To prove Theorem 15, we use again an inductive approach very similar to that from [MS15].

Proof. By symmetry, we may assume that $k \leq \lfloor (n-1)/2 \rfloor$. We prove the statement by induction on n .

For any $n \geq 3$ and $k = 0$ such a tight enumeration trivially exists (see Theorem 8 (i)). Moreover, as the vertex 0^n is adjacent to all vertices in level 1, this enumeration must contain a switched 2-path. Note also that the statement holds for all odd $n \geq 3$ and $k = \lfloor n/2 \rfloor$ by Theorem 1. Indeed, since the corresponding enumeration is a Hamilton cycle in $Q_{n,[k,k+1]}$, it must contain a 3-path.

These observations settle in particular the base cases $n = 3$, $k \in \{0, 1\}$.

For the induction step we assume that the statement holds for some $n \geq 3$ and all relevant values of k , and prove that it also holds for $n+1$ and all $1 \leq k < \lfloor n/2 \rfloor$. In any level i of Q_n , we consider two special vertices $a_{n,i}$ and $b_{n,i}$ as defined in (18). By induction, there is a tight enumeration C_0 of the vertices of $Q_{n,[k,k+1]}$ that contains a 3-path (note that $k \geq 1$). We also know that there is a tight enumeration C_1 of the vertices of $Q_{n,[k-1,k]}$ that contains a switched 2-path (note that $k-1 < \lfloor n/2 \rfloor$). In C_0 we permute coordinates so that the 3-path is mapped to $(a_{n,k}, a_{n,k+1}, b_{n,k}, b_{n,k+1})$. Moreover, in C_1 we permute coordinates so that the switched 2-path is mapped to $(b_{n,k-1}, b_{n,k}, a_{n,k})$ (this is possible because $(b_{n,k}, b_{n,k-1}, a_{n,k})$ is a path of length 2 in Q_n).

To make the description of the following modifications easier, let us think about C_0 and C_1 as cycles in $Q_{n,[k,k+1]}$ and $Q_{n,[k-1,k]}$ that use some additional distance-2 edges (not actually present in Q_n) between vertices in the upper levels $k+1$ or k , respectively. From C_0 we remove the distance-1 edge $(b_{n,k}, a_{n,k+1})$ and attach a 0-bit to all vertices. From C_1 we remove the distance-2 edge $(a_{n,k}, b_{n,k})$ and attach a 1-bit to all vertices. We connect the resulting paths by adding the distance-2 edge $(a_{n,k+1} \circ 0, a_{n,k} \circ 1)$ and the distance-1 edge $(b_{n,k} \circ 0, b_{n,k} \circ 1)$. In this way, we clearly obtain an enumeration C of all vertices of $Q_{n+1,[k,k+1]}$. Moreover, all distance-2 steps of C are within the same bipartite class (in level $k+1$), and the remaining steps are distance-1 steps, implying that C is a tight enumeration. Furthermore, $(b_{n,k-1} \circ 1, b_{n,k} \circ 1, b_{n,k} \circ 0, b_{n,k+1} \circ 0)$ is a 3-path in C , and $(a_{n,k} \circ 0, a_{n,k+1} \circ 0, a_{n,k} \circ 1)$ is a switched 2-path. This completes the proof. \square

Proof of Theorem 8 (iia). Follows immediately from Theorem 15. \square

The proof of Theorem 8 (iiib) is analogous to the proof of Theorem 7 (iii), and it proceeds by gluing together several cycles obtained from Theorem 15. We include the proof for the sake of completeness, as there are some minor technical differences. In particular, to make the proof work we will need the 3-path and the switched 2-path guaranteed by Theorem 15.

Proof of Theorem 8 (iiib). We only consider the case $1 \leq k < l \leq \lceil n/2 \rceil$, the other case follows by symmetry. By Theorem 15, there is a tight enumeration C_i of all vertices in $Q_{n,[i,i+1]}$ for every $i = k, k+2, \dots, l-1$. The only distance-2 steps of each C_i are in level $i+1$ since $l \leq \lceil n/2 \rceil$. It is again useful to think of C_i as a cycle in $Q_{n,[i,i+1]}$ that uses some additional distance-2 edges (not actually present in Q_n) between vertices in the upper level $i+1$. We now show how to join these $(l-k+1)/2 \geq 2$ cycles to a single cycle that enumerates all vertices of $Q_{n,[k,l]}$. In any level i of Q_n , we consider two special vertices $a_{n,i} = a_i$ and $b_{n,i} = b_i$ as defined in (18).

In the lowest cycle C_k between levels k and $k+1$ we permute coordinates so that the switched 2-path is mapped to (b_k, b_{k+1}, a_{k+1}) . Furthermore, in each of the other cycles C_i , $i = k+2, k+4, \dots, l-1$, we permute coordinates (independently for each cycle) so that the 3-path is mapped to $(a_i, a_{i+1}, b_i, b_{i+1})$. We now modify the (permuted) cycles as follows: In the lowest cycle C_k we remove the distance-2 edge (b_{k+1}, a_{k+1}) . In the intermediate cycles C_i , $i = k+2, k+4, \dots, l-3$, we remove the distance-1 edges (a_i, a_{i+1}) and (b_i, b_{i+1}) . In the uppermost cycle C_{l-1} , we remove the distance-1 edges (a_{l-1}, a_l) and (b_{l-1}, b_l) , and we add the distance-2 edge (a_l, b_l) . Finally, we connect the resulting paths by

adding the distance-1 edges (a_{i-1}, a_i) and (b_{i-1}, b_i) for $i = k+2, k+4, \dots, l-1$. In this way, we obtain an enumeration C of all vertices of $Q_{n,[k,l]}$. Moreover, all distance-2 steps of C are within the same bipartite class (in levels $k+1, k+3, \dots, l$), and the remaining steps are distance-1 steps, implying that C is a tight enumeration. \square

Proof of Theorem 8 (iv). This proof works analogously to the proof of Theorem 7 (iv) by induction over the dimension of the cube. As in the proof of Theorem 15, we join pairs of ‘cycles’ along a 3-path and a switched 2-path. We leave the details to the reader. \square

4. EFFICIENT ALGORITHMS

In the following we first state some known facts about the reflected Gray code Γ_n , which will allow us to formulate a loopless algorithm to generate trimmed Gray codes, proving part (a) of Theorem 9 (algorithms for cases (i) and (ii) of Theorems 7 and 8). We finally describe an efficient algorithm for glued Gray codes, proving part (b) of Theorem 9 (algorithms for case (iii) of Theorems 7 and 8).

4.1. Explicit description of trimmed Gray codes. Let x be a vertex of Q_n in level k . It is well-known [Knu11, Section 7.2.1.1] that the successor of x in Γ_n is given by

$$s_{\Gamma_n}(x) = \begin{cases} x \oplus e_1 & \text{if } k \text{ is even} , \\ x \oplus e_{i+1} & \text{if } k \text{ is odd} , \end{cases} \quad (20)$$

where $i \geq 1$ is the smallest integer with $x_i = 1$ (except for $x = \ell(\Gamma_n) = 0^{n-1}1$ where we set $i := n-1$) and $e_j := 0^{j-1}10^{n-j}$ for all $j = 1, 2, \dots, n$.

From (20) we immediately obtain the following characterization of upward vertices (and consequently also of downward vertices) in Γ_n .

Lemma 16. *The vertex $x = (x_1, x_2, \dots, x_n)$ in level k of Q_n is an upward vertex in Γ_n if and only if k is even and $x_1 = 0$ or k is odd and $x_{i+1} = 0$, where i is as defined in (20).*

For any $0 \leq k \leq n$ we let $u_{n,k}$ and $d_{n,k}$, respectively, denote the number of upward or downward vertices of Γ_n in level k of Q_n . Recall that by Lemma 13 all vertices in the sequences $\text{up}(\Gamma_{n,k})$ and $\text{down}(\Gamma_{n,k})$ are distinct, and therefore the length of these sequences is given by $u_{n,k}$ and $d_{n,k}$, respectively. From Lemma 16 we obtain that

$$u_{n,k} = \binom{n-1}{k} \quad \text{and} \quad d_{n,k} = \binom{n}{k} - \binom{n-1}{k} = \binom{n-1}{k-1} . \quad (21)$$

Next, we state explicit descriptions for the successor of a vertex in $\Gamma_{n,k}$ given by Tang and Liu, see [TL73, Theorem 4]. Note that we use a slightly different notation with the most significant bit at the last position.

Lemma 17 ([TL73]). *Let $n \geq 1$ and $0 < k < n$ and let x be an upward vertex in level k of Q_n . Then we have $s_{\Gamma_{n,k}}(x) = x \oplus e_{p_1} \oplus e_{p_2}$ and $\text{up}(x, s(x)) = x \oplus e_{p_1}$ where*

$$p_1 = \begin{cases} i-1 & \text{if } k \text{ is even} \\ i+1 & \text{if } k \text{ is odd} \end{cases} , \quad p_2 = i ,$$

and $i \geq 1$ is the smallest integer with $x_i = 1$.

Combining (20) and Lemma 17 yields that for any upward vertex x in level k , $0 < k < n$, and for $y := s_{\Gamma_{n,k}}(x)$ we have

$$\text{up}(x, y) = \begin{cases} p_{\Gamma_n}(y) & \text{if } k \text{ is even} , \\ s_{\Gamma_n}(x) & \text{if } k \text{ is odd} , \end{cases} \quad (22)$$

a further strengthening of Lemma 14 (and of the first part of Lemma 13).

For downward vertices, the description of successors in $\Gamma_{n,k}$, given by the next lemma, is more complicated. The reason is that successors of some downward vertices change when we construct $\Gamma_{n+1,k}$ from $\Gamma_{n,k}$ and $\Gamma_{n,k-1}$ as described by (2b), recall the proof of Lemma 13. For the same reason, unfortunately the property (22) (or an analogous statement) does not hold for downward vertices.

Lemma 18 ([TL73]). *Let $n \geq 1$ and $0 < k < n$ and let x be a downward vertex in level k of Q_n . Then we have $s_{\Gamma_{n,k}}(x) = x \oplus e_{p_1} \oplus e_{p_2}$ and $\text{down}(x, s(x)) = x \oplus e_{p_1}$ where*

$$\begin{aligned} p_1 &= i - 2 \text{ and } p_2 = i && \text{if } k \not\equiv i \pmod{2} , \\ p_1 &= n \text{ and } p_2 = i && \text{if } k \equiv i \pmod{2} \text{ and } j = n , \\ p_1 &= i - 1 \text{ and } p_2 = j + 1 && \text{if } k \equiv i \pmod{2}, j < n \text{ and } x_{j+1} = 0 , \\ p_1 &= j + 1 \text{ and } p_2 = i && \text{if } k \equiv i \pmod{2}, j < n \text{ and } x_{j+1} = 1 , \end{aligned}$$

$i \geq 1$ is the smallest integer with $x_i = 0$, and $j > i$ is the smallest integer with $x_j = 1$.

Using (20) and Lemmas 16–18 we are now ready to derive a loopless algorithm for generating trimmed Gray codes.

4.2. A loopless algorithm for trimmed Gray codes. Loopless algorithms both for the reflected Gray code Γ_n and for its restriction $\Gamma_{n,k}$ to one level of the cube (i.e., an algorithmic version of Theorem 6) were already provided in [Ehr73, BER76]. However, these two algorithms cannot simply be merged into a loopless algorithm producing the trimmed Gray code. Instead, we provide a loopless algorithm that is based on the explicit description of successors given in the previous section.

Consider now the algorithm $\text{TRIMGC}(n, k, l)$ that computes a trimmed Gray code between levels k and l with $l - k \geq 2$ of Q_n as described in Section 2.1 (Theorem 11), i.e., the algorithm produces a saturating cycle for cases (i) and (ii) of Theorem 7. At the end of this section we describe how to modify the algorithm to produce a tight enumeration for cases (i) and (ii) of Theorem 8.

The algorithm maintains the current vertex in the variable x and the current level in the variable c , $k < c < l$. Both are initialized in line T1 to $c = k + 1$ and $x = 1^c 0^{n-c}$ (the code can easily be modified to start at a different vertex). The algorithm visits the sequence of vertices along the trimmed Gray code by the calls $\text{VISIT}(x)$, which could be some user-defined function that reads x . For simplicity the main while-loop does not have an explicit termination criterion, but this can be added easily (e.g., by providing an additional argument to the algorithm that specifies the number of vertices to be visited before termination). If the while-loop is not terminated, then the same cycle is traversed over and over again. There are four main cases distinguished in the while-loop of the algorithm: If the current level is between $k + 1$ and $l - 1$, then we either follow Γ_n by flipping a 0-bit to a 1-bit (if the condition in line T5 is satisfied), or we follow Γ_n by flipping a 1-bit to a 0-bit (if the condition in line T12 is satisfied). If the current level is $c = l - 1$ and x is an upward vertex (the condition in line T19 is satisfied), then we first flip a 0-bit to the vertex $\text{up}(x, y)$, $y := s_{\Gamma_{n,c}}(x)$, and then a 1-bit to the vertex y . On the other hand, if the current level is $c = k + 1$ and x is a downward vertex (the condition in line T26 is satisfied), then we first flip a 1-bit to the vertex $\text{down}(x, y)$, $y := s_{\Gamma_{n,c}}(x)$, and then a 0-bit to the vertex y .

The key to our loopless algorithm is to be able to determine in constant time the smallest integer $i \geq 1$ with $x_i = 1$ or with $x_i = 0$, recall (20) and Lemmas 16–18. Furthermore, in Lemma 18 we also need the smallest integer $j > i$ with $x_j = 1$. To achieve this the algorithm maintains the following data structures: We maintain an array (p_1, p_2, \dots, p_c) with the positions of the 1-bits in $x = (x_1, x_2, \dots, x_n)$ where p_i , $1 \leq i \leq c$, is the position of the i -th 1-bit in x counted from the right (from the highest index n). Thus p_c is the position of the first 1-bit in x from the left (from the

Algorithm 1: TRIMGC(n, k, l)

Input: Integers $n \geq 2$ and $0 \leq k < l \leq n$, $l - k \geq 2$.

Result: The algorithm visits all vertices of the trimmed Gray code in $Q_{n,[k,l]}$ (which is a saturating cycle in cases (i) and (ii) of Theorem 7).

```

T1  $c := k + 1; x := 1^c 0^{n-c};$  /* initialize current level  $c$  and current vertex  $x$  */
T2  $\nu_c := 1;$  for  $i := 1$  to  $c$  do  $p_i := c - i + 1$  /* initialize  $(\nu_1, \dots, \nu_c)$  and  $(p_1, \dots, p_c)$  */
T3 while not enough vertices visited do
T4   if  $(c \text{ is even} \wedge x_1 = 0) \vee (c \text{ is odd} \wedge p_c < n \wedge x_{p_c+1} = 0)$  then  $\text{up} := \text{true}$  else  $\text{up} := \text{false}$ 
T5   if  $(\text{up} = \text{true} \wedge c < l - 1)$  then /* follow  $\Gamma_n$  up */
T6     if  $c$  is even then /*  $x_1 = 0$  */
T7        $x_1 := 1; \text{VISIT}(x); c := c + 1; p_c := 1$ 
T8       if  $x_2 = 0$  then  $\nu_c := c$  else  $\nu_c := \nu_{c-1}$ 
T9     else /*  $c$  is odd and  $x_{i+1} = 0$  */
T10       $i := p_c; x_{i+1} := 1; \text{VISIT}(x); c := c + 1; p_c := i; p_{c-1} := i + 1$ 
T11      if  $(i + 1 = n \vee x_{i+2} = 0)$  then  $\nu_c := c - 1$  else  $\nu_c := \nu_{c-2}$ 
T12   else if  $(\text{up} = \text{false} \wedge c > k + 1)$  then /* follow  $\Gamma_n$  down */
T13     if  $c$  is even then /*  $x_1 = 1$  */
T14        $x_1 := 0; \text{VISIT}(x); c := c - 1$ 
T15       if  $x_2 = 1$  then  $\nu_c := \nu_{c+1}$ 
T16     else /*  $c$  is odd and  $x_{i+1} = 1$  */
T17        $i := p_c; x_{i+1} := 0; \text{VISIT}(x); c := c - 1; p_c := i; \nu_c := c$ 
T18       if  $x_{i+2} = 1$  then  $\nu_{c-1} := \nu_{c+1}$ 
T19   else if  $(\text{up} = \text{true} \wedge c = l - 1)$  then /* follow  $\Gamma_{n,c}$  through  $\text{up}(x, s_{\Gamma_{n,c}}(x))$  */
T20     if  $c$  is even then /*  $x_{i-1} = 0$  and  $x_i = 1$  */
T21        $i := p_c; x_{i-1} := 1; \text{VISIT}(x); x_i := 0; \text{VISIT}(x); p_c := i - 1$ 
T22       if  $x_{i+1} = 1$  then  $\nu_{c-1} := \nu_c; \nu_c := c$ 
T23     else /*  $c$  is odd,  $x_{i+1} = 0$  and  $x_i = 1$  */
T24        $i := p_c; x_{i+1} := 1; \text{VISIT}(x); x_i := 0; \text{VISIT}(x); p_c := i + 1$ 
T25       if  $(i + 2 \leq n \wedge x_{i+2} = 1)$  then  $\nu_c := \nu_{c-1}$ 
T26   else /*  $\text{up} = \text{false} \wedge c = k + 1$ ; follow  $\Gamma_{n,c}$  through  $\text{down}(x, s_{\Gamma_{n,c}}(x))$  */
T27     if  $x_1 = 0$  then  $i := 1$  else  $i := p_{\nu_c} + 1$  /*  $i$  is minimal s.t.  $x_i = 0$  */
T28     if  $c \not\equiv i \pmod 2$  then /*  $x_{i-2} = 1$  and  $x_i = 0$  */
T29        $x_{i-2} := 0; \text{VISIT}(x); x_i := 1; \text{VISIT}(x); p_{\nu_c+1} := i - 1; p_{\nu_c} := i$ 
T30       if  $(i + 1 \leq n \wedge x_{i+1} = 1)$  then  $\nu_{\nu_c+1} := \nu_{\nu_c-1}$  else  $\nu_{\nu_c+1} := \nu_c$ 
T31       if  $i > 3$  then  $\nu_c := \nu_c + 2$ 
T32     else /*  $c \equiv i \pmod 2$ ;  $j > i$  is minimal s.t.  $x_j = 1$  */
T33       if  $x_1 = 0$  then  $a := c; j := p_a$  else  $a := \nu_c - 1; j := p_a$ 
T34       if  $j = n$  then  $x_n := 0; \text{VISIT}(x); x_i := 1; \text{VISIT}(x); p_1 := i; \nu_c := 1$ 
T35       else if  $x_{j+1} = 0$  then /*  $j < n$ ,  $x_{i-1} = 1$  and  $x_{j+1} = 0$  */
T36          $x_{i-1} := 0; \text{VISIT}(x); x_{j+1} := 1; \text{VISIT}(x); p_{\nu_c} := j; p_{\nu_c-1} := j + 1$ 
T37         if  $(j + 2 \leq n \wedge x_{j+2} = 1)$  then  $\nu_{\nu_c} := \nu_{\nu_c-2}$  else  $\nu_{\nu_c} := \nu_c - 1$ 
T38         if  $i > 2$  then  $\nu_c := \nu_c + 1$ 
T39       else /*  $j < n$ ,  $x_{j+1} = 1$  and  $x_i = 0$  */
T40          $x_{j+1} := 0; \text{VISIT}(x); x_i := 1; \text{VISIT}(x); p_a := i; p_{a-1} := j$ 
T41         if  $(j + 2 \leq n \wedge x_{j+2} = 1)$  then  $\nu_{a-2} := \nu_a$ 
T42         if  $j = i + 1$  then  $\nu_c := a - 1$  else  $\nu_{a-1} := a - 1, \nu_c := a$ 

```

lowest index 1). Since adding and removing 1's happens around the position p_c (recall (20)), the length of this array changes dynamically. For $i > c$ the value of p_i is undefined (the algorithm does not ‘clean up’ those values after using them). We also maintain an array $(\nu_1, \nu_2, \dots, \nu_c)$ to quickly find the smallest integer $j > i$ with $x_j = 0$ where i is the smallest integer with $x_i = 1$. However, the value of ν_i , $1 \leq i \leq c$, is defined only if p_i is a starting position of a substring of 1-bits in x ; i.e., $x_{p_i} = 1$ and $x_{p_i-1} = 0$, or $p_i = 1$. In this case, the value of ν_i is the index such that p_{ν_i} is the position where the corresponding substring of 1-bits ends in x . In particular, since p_c is the position of the first 1 in x , the value $p_{\nu_c} + 1$ is the smallest integer greater than p_c such that x has a 0-bit at this position. Initially, there is only one substring of 1-bits in $x = 1^c 0^{n-c}$ that starts at position $p_c = 1$ and ends at position $p_1 = c$, so we have $\nu_c = 1$ (see line T2).

4.2.1. Correctness of the algorithm. We now argue about the correctness of the algorithm $\text{TRIMGC}(n, k, l)$. In lines T5 and T12 (see also line T4), by Lemma 16 the algorithm correctly checks whether x is an upward vertex in a level $c \in [k+1, l-2]$ or a downward vertex in a level $c \in [k+2, l-1]$, respectively. The correctness of the corresponding modifications in lines T6–T11 and lines T13–T18 follows immediately from (20). In lines T20–T25 the algorithm moves from x in level $c = l-1$ to $s_{\Gamma_{n,c}}(x) =: y$ via the common neighbor $\text{up}(x, y)$ in level l , which is correct by Lemma 17. Similarly, in lines T27–T42 the algorithm moves from x in level $c = k+1$ to $s_{\Gamma_{n,c}}(x) =: y$ via the common neighbor $\text{down}(x, y)$ in level k , which is correct by Lemma 18. It can be verified straightforwardly that the algorithm correctly updates all relevant entries of p and ν in all cases.

4.2.2. Running time and space requirements of the algorithm. We first argue about the running time of the algorithm $\text{TRIMGC}(n, k, l)$. Clearly, the initialization in lines T1 and T2 takes time $\mathcal{O}(n)$. In all subsequent steps, there are only a constant number of operations between any two $\text{VISIT}(x)$ calls, so the algorithm is indeed loopless (each bitstring is generated in time $\mathcal{O}(1)$). The space required by our algorithm is $\mathcal{O}(n)$, as each of the arrays x , p and ν has at most n entries.

4.2.3. Proof of part (a) of Theorem 9. The previous arguments show that the algorithm $\text{TRIMGC}(n, k, l)$ correctly produces a saturating cycle in $Q_{n,[k,l]}$ in the claimed running time for cases (i) and (ii) of Theorem 7. To obtain a loopless algorithm that generates a tight enumeration of the vertices of $Q_{n,[k,l]}$, we simply call $\text{TRIMGC}(n, k-1, l+1)$ and omit the first of the two $\text{VISIT}(x)$ calls in each of the lines T21, T24, T29, T34, T36 and T40. The algorithm then moves directly with a distance-2 step from a vertex x in level k or l to the vertex $s_{\Gamma_{n,k}}(x)$ or $s_{\Gamma_{n,l}}(x)$, respectively, without visiting $\text{down}(x, s_{\Gamma_{n,k}}(x))$ or $\text{up}(x, s_{\Gamma_{n,l}}(x))$ in between. Also, if $k = 0$ then line T2 has to be omitted, and the special case $c = 1$ and $x = 0^{n-1}1$ mentioned after (20) must be treated separately in lines T17 and T18. This proves part (a) of Theorem 9.

4.3. Efficient algorithms for glued Gray codes. In this section we present an algorithm $\text{SATCYCLE}(n, k)$ that computes a saturating cycle in $Q_{n,[k,k+1]}$, an algorithmization of Theorem 3 (the existence of such a cycle was first proved in [MS15]). This algorithm forms the basis for all further algorithms to produce saturating cycles and tight enumerations across an even number of levels of the cube (recall the gluing technique from the proofs of Theorem 7 (iii) and Theorem 8 (iiia) and (iiib)). For space reasons we do not explicitly specify these more general algorithms as pseudocode in our paper, but rather explain how they can be derived from the algorithm $\text{SATCYCLE}(n, k)$. All algorithms are implemented in the C++ code we provide with this paper, to be found on our website [www] (so implementation details can be looked up there).

The algorithm $\text{SATCYCLE}(n, k)$ is essentially a recursive implementation of the inductive proof of Theorem 3 (induction over n) given in [MS15]. It starts at the vertex $x := a_{n,k}$ (line S1, recall the definition (18)) and it uses a stack S (initialized in line S2) to keep track of the recursion steps of the computation (thereby avoiding any recursive calls). There are two different types of items

Algorithm 2: SATCYCLE(n, k)

Input: Integers $n \geq 3$ and $1 \leq k \leq \lfloor (n-1)/2 \rfloor$
Result: A saturating cycle in $Q_{n,[k,k+1]}$

```

S1  $x := a_{n,k}$                                 /* initialize starting vertex */
S2 stack  $S$                                     /* initialize empty stack  $S$  */
S3  $S.PUSH((\text{rec}, (n, k, \rightarrow)))$           /* (3) visit  $b_{n,k} \rightarrow a_{n,k}$  without  $a_{n,k+1}$  */
S4  $S.PUSH((\text{flip}, n))$                         /* (2)  $a_{n,k+1} \rightarrow b_{n,k}$  */
S5  $S.PUSH((\text{flip}, n - k))$                     /* (1)  $a_{n,k} \rightarrow a_{n,k+1}$  */
S6 while  $S.EMPTY() = \text{false}$  do
S7    $(\text{instr}, r) := S.POP()$                   /* pop from the stack */
S8   if  $\text{instr} = \text{flip}$  then                  /* popped a flip-item, perform one bitflip */
S9      $i := r$ 
S10     $x_i := 1 - x_i$ 
S11     $VISIT(x)$ 
S12  else                                    /* popped a rec-item, perform one recursion step */
S13     $(n, k, \text{dir}) := r$ 
S14    if  $k \geq 1 \wedge n = 2k + 1$  then          /* middle levels conjecture cycle */
S15      if  $\text{dir} = \leftarrow$  then
S16         $HAMCYCLE(k, x, \leftarrow)$           /* visit  $a_{n,k} \rightarrow b_{n,k}$  without  $a_{n,k+1}$  */
S17      else
S18         $HAMCYCLE(k, x, \rightarrow)$         /* visit  $b_{n,k} \rightarrow a_{n,k}$  without  $a_{n,k+1}$  */
S19    else if  $k \geq 1 \wedge n > 2k + 1$  then
S20      if  $\text{dir} = \leftarrow$  then              /* visit  $a_{n,k} \rightarrow b_{n,k}$  without  $a_{n,k+1}$  */
S21         $S.PUSH((\text{rec}, (n-1, k, \rightarrow)))$  /* (4)  $b_{n-1,k} \circ 0 \rightarrow a_{n-1,k} \circ 0 = b_{n,k}$  */
S22         $S.PUSH((\text{flip}, n))$                 /* (3)  $b_{n-1,k} \circ 1 \rightarrow b_{n-1,k} \circ 0$  */
S23         $S.PUSH((\text{flip}, n - k - 1))$         /* (2)  $b_{n-1,k-1} \circ 1 \rightarrow b_{n-1,k} \circ 1$  */
S24         $S.PUSH((\text{rec}, (n-1, k-1, \leftarrow)))$  /* (1)  $a_{n,k} = a_{n-1,k-1} \circ 1 \rightarrow b_{n-1,k-1} \circ 1$  */
S25      else                                /* visit  $b_{n,k} \rightarrow a_{n,k}$  without  $a_{n,k+1}$  */
S26         $S.PUSH((\text{rec}, (n-1, k-1, \rightarrow)))$  /* (4)  $b_{n-1,k-1} \circ 1 \rightarrow a_{n-1,k-1} \circ 1 = a_{n,k}$  */
S27         $S.PUSH((\text{flip}, n - k - 1))$         /* (3)  $b_{n-1,k} \circ 1 \rightarrow b_{n-1,k-1} \circ 1$  */
S28         $S.PUSH((\text{flip}, n))$                 /* (2)  $b_{n-1,k} \circ 0 \rightarrow b_{n-1,k} \circ 1$  */
S29         $S.PUSH((\text{rec}, (n-1, k, \leftarrow)))$  /* (1)  $b_{n,k} = a_{n-1,k} \circ 0 \rightarrow b_{n-1,k} \circ 0$  */

```

pushed and popped from the stack, which we call **flip-item** and **rec-items**. A **flip-item** is a pair (flip, i) , where the variable i , $1 \leq i \leq n$, indicates the bit position in the current vertex x to be flipped in some step. Such items are pushed onto the stack S in lines S4, S5, S22, S23, S27 and S28. A **rec-item** is a pair $(\text{rec}, (n', k', \text{dir}))$, $\text{dir} \in \{\leftarrow, \rightarrow\}$, where the variables n' and k' , $3 \leq n' \leq n$, $1 \leq k' \leq \lfloor (n'-1)/2 \rfloor$, indicate the recursion step to compute a saturating path in $Q_{n',[k',k'+1]}$, and the variable **dir** indicates the direction in which this path is to be traversed. By a *saturating path* we mean a path that visits all vertices on level k' , and that starts and ends on this level. Specifically, if $\text{dir} = \leftarrow$ then the saturating path in $Q_{n',[k',k'+1]}$ will be traversed starting at the vertex $a_{n',k'}$ and ending at the vertex $b_{n',k'}$, and it will omit the vertices $a_{n',k'+1}$ and $b_{n',k'+1}$ (so to complete this path to a saturating cycle, only the vertex $a_{n',k'+1}$ has to be added). If $\text{dir} = \rightarrow$ then the start and end vertices of the path are interchanged (the path starts at $b_{n',k'}$ and ends at $a_{n',k'}$). Such

rec-items are pushed onto the stack S in lines S3, S21, S24, S26 and S29. Note that the values of n and k of the algorithm $\text{SATCYCLE}(n, k)$ are overwritten in line S13 with the values n' and k' of **rec**-items previously pushed onto the stack.

In each iteration of the main while-loop of the algorithm, one item is popped from the stack (line S7). For every **flip**-item popped from the stack, the corresponding bit is flipped in x (lines S9–S11). For every **rec**-item ($\text{rec}, (n, k, \text{dir})$), $\text{dir} \in \{\leftarrow, \rightarrow\}$, popped from the stack, there are three cases to consider: If $k = 0$, then we do nothing. This is captured by a non-existent else-branch of the if-statement in line S14. If $k \geq 1$ and $n > 2k + 1$, then the **rec**-item is replaced by two **rec**-items and two **flip**-items (lines S20–S29), i.e., we perform a recursion step to construct a saturating path in $Q_{n,[k,k+1]}$ by gluing together two saturating paths in $Q_{n-1,[k,k+1]}$ and $Q_{n-1,[k-1,k]}$ (in the special case $k = 1$, the latter path is degenerate). The gluing that joins the two saturating paths to one is achieved by two bitflips enforced by the two **flip**-items that are pushed onto the stack. The exact details how this gluing is achieved will become clear in the next section. Note however that in order to achieve a certain order of bitflips, the corresponding items have to be pushed onto the stack in reverse order. The later execution order is indicated on the right-hand side of our pseudocode in the comments, indicated by numbers (1)–(3) and (1)–(4) in lines S3–S5, S21–S24 and S26–S29. In the third case, if $k \geq 1$ and $n = 2k + 1$, then we encounter a nontrivial boundary case of our recursion (the middle levels conjecture, Theorem 1). This case is handled by calling the constant-time algorithm $\text{HAMCYCLE}()$ for computing a Hamilton cycle in the graph $Q_{2k+1,[k,k+1]}$ presented in [MN15, MN16]. We modify this algorithm (by applying suitable bit permutations) so that it starts at the vertex $a_{2k+1,k}$ and ends at the vertex $b_{2k+1,k}$ or vice-versa, and so that it visits all vertices of $Q_{2k+1,[k,k+1]}$ except the first vertex $a_{2k+1,k}$ or $b_{2k+1,k}$, respectively, and except the vertex $a_{2k+1,k+1}$ (to make this saturating path a Hamilton cycle, only this vertex has to be added). We call the corresponding algorithms $\text{HAMCYCLE}(k, x, \leftarrow)$ (start at $a_{2k+1,k}$ and end at $b_{2k+1,k}$) and $\text{HAMCYCLE}(k, x, \rightarrow)$ (start at $b_{2k+1,k}$ and end at $a_{2k+1,k}$), respectively, where the bitstring x is an additional argument to which all internally computed bitflips (by the algorithm $\text{HAMCYCLE}()$) are applied, followed by corresponding $\text{VISIT}(x)$ calls (all this happens inside $\text{HAMCYCLE}()$). Possibly $2k + 1$ is strictly smaller than the length of x , but within the two $\text{HAMCYCLE}(k, x, \text{dir})$ calls, $\text{dir} \in \{\leftarrow, \rightarrow\}$, in line S16 and S18, only the first $2k + 1$ bits of x are modified.

4.3.1. Correctness of the algorithm. The next lemma states that the algorithm $\text{SATCYCLE}(n, k)$ correctly glues together several saturating cycles of smaller dimension in the graphs $Q_{n',[k',k'+1]}$, where $3 \leq n' \leq n$ and $1 \leq k' \leq \lfloor (n' - 1)/2 \rfloor$, to a single saturating cycle in $Q_{n,[k,k+1]}$ (visiting all the corresponding vertices along the way).

To state the lemma, we write $B_{n,k}$ for all bitstrings of length n with weight k . For $x = (x_1, x_2, \dots, x_n)$ and $1 \leq i, j \leq n$ we write $x_{[i,j]} := (x_i, x_{i+1}, \dots, x_j)$ (in the degenerate case $i > j$ we have $x_{[i,j]} = ()$). Moreover, we say that an item on the stack S of our algorithm *has been processed* at the moment the item below it is popped from the stack. The item at the bottom of the stack S *has been processed* when the algorithm terminates.

Lemma 19. *The algorithm $\text{SATCYCLE}(n, k)$ satisfies the following invariant for all inputs $n \geq 3$ and $1 \leq k \leq \lfloor (n - 1)/2 \rfloor$:*

- (i) *At the moment a **rec**-item ($\text{rec}, (n', k', \leftarrow)$) is popped from the stack S , the current vertex x satisfies $x_{[1,n']} = a_{n',k'}$, and when this item has been processed, the current vertex x satisfies $x_{[1,n']} = b_{n',k'}$. Moreover, in between all vertices of the form $x' \circ x_{[n'+1,n]}$, $x' \in B_{n',k'}$, except the vertex $a_{n',k'} \circ x_{[n'+1,n]}$ are visited, no vertex is visited twice. Furthermore, the vertex $a_{n',k'+1} \circ x_{[n'+1,n]}$ is not visited, and if $n' > 2k' + 1$, then the vertex $b_{n',k'+1} \circ x_{[n'+1,n]}$ is not visited either. Put differently, $a_{n',k'} \circ x_{[n'+1,n]}$ together with the visited vertices forms a saturating path in $Q_{n',[k',k'+1]} \circ x_{[n'+1,n]}$.*

- (ii) At the moment a **rec**-item $(\text{rec}, (n', k', \rightarrow))$ is popped from the stack S , the current vertex x satisfies $x_{[1, n']} = b_{n', k'}$, and when this item has been processed, the current vertex x satisfies $x_{[1, n']} = a_{n', k'}$. Moreover, in between all vertices of the form $x' \circ x_{[n'+1, n]}$, $x' \in B_{n', k'}$, except the vertex $b_{n', k'} \circ x_{[n'+1, n]}$ are visited, no vertex is visited twice. Furthermore, the vertex $b_{n', k'+1} \circ x_{[n'+1, n]}$ is not visited, and if $n' > 2k' + 1$, then the vertex $b_{n', k'+1} \circ x_{[n'+1, n]}$ is not visited either. Put differently, $b_{n', k'} \circ x_{[n'+1, n]}$ together with the visited vertices forms a saturating path in $Q_{n', [k', k'+1]} \circ x_{[n'+1, n]}$.

Proof. To prove the lemma we consider the recursion tree $T = T(n, k)$ corresponding to the algorithm $\text{SATCYCLE}(n, k)$. The tree T is defined as follows (see Figure 6): The tree is rooted and it is a binary tree, i.e., each node has either a left and a right child, or no children at all (then it is a leaf). The nodes of T are the **rec**-items that are stored on the stack S in the course of the algorithm, and the **rec**-item $(\text{rec}, (n, k, \rightarrow))$ that is pushed onto the stack S in the first step (line S3) is the root. The remaining nodes of T are defined recursively: Each node $(\text{rec}, (n', k', \text{dir}))$, $\text{dir} \in \{\leftarrow, \rightarrow\}$, satisfying $k' = 0$, or $k' \geq 1$ and $n' = 2k' + 1$ is a leaf. Otherwise, if $\text{dir} = \leftarrow$, then the node has the left child $(\text{rec}, (n' - 1, k' - 1, \leftarrow))$ and the right child $(\text{rec}, (n' - 1, k', \rightarrow))$, and if $\text{dir} = \rightarrow$, then the node has the left child $(\text{rec}, (n' - 1, k', \leftarrow))$ and the right child $(\text{rec}, (n' - 1, k' - 1, \rightarrow))$. By the instructions in lines S21–S24 and lines S26–S29, left children correspond to **rec**-items located at the top of the stack S (at the end of the iteration of the while-loop in which they are pushed onto S), whereas right children are **rec**-items located three items below the top (at this moment). Note that several nodes of T may have the same signature $(\text{rec}, (n', k', \text{dir}))$ (see Figure 6), but these are different **rec**-items encountered on the stack S in the course of the algorithm.

To prove the lemma we prove the following auxiliary statements:

- (1) At the moment the item $(\text{rec}, (n, k, \rightarrow))$ is popped from the stack, the current vertex x satisfies $x = b_{n, k}$.
- (2) Given a non-leaf node $(\text{rec}, (n', k', \leftarrow))$ and its left child $(\text{rec}, (n' - 1, k' - 1, \leftarrow))$ in T , suppose that at the moment the item $(\text{rec}, (n', k', \leftarrow))$ is popped from the stack the current vertex x satisfies $x_{[1, n']} = a_{n', k'}$. Then at the moment the left child $(\text{rec}, (n' - 1, k' - 1, \leftarrow))$ is popped from the stack the current vertex x satisfies $x_{[1, n'-1]} = a_{n'-1, k'-1}$.
- (3) Given a non-leaf node $(\text{rec}, (n', k', \rightarrow))$ and its left child $(\text{rec}, (n' - 1, k', \leftarrow))$ in T , suppose that at the moment the item $(\text{rec}, (n', k', \rightarrow))$ is popped from the stack the current vertex x satisfies $x_{[1, n']} = b_{n', k'}$. Then at the moment the left child $(\text{rec}, (n' - 1, k', \leftarrow))$ is popped from the stack the current vertex x satisfies $x_{[1, n'-1]} = a_{n'-1, k'}$.
- (4) Given a non-leaf node $(\text{rec}, (n', k', \leftarrow))$ and its right child $(\text{rec}, (n' - 1, k', \rightarrow))$ in T , suppose that at the moment the item $(\text{rec}, (n' - 1, k' - 1, \leftarrow))$ (this is the left child) has been processed the current vertex x satisfies $x_{[1, n'-1]} = b_{n'-1, k'-1}$. Then at the moment the right child $(\text{rec}, (n' - 1, k', \rightarrow))$ is popped from the stack the current vertex x satisfies $x_{[1, n'-1]} = b_{n'-1, k'}$.
- (5) Given a non-leaf node $(\text{rec}, (n', k', \rightarrow))$ and its right child $(\text{rec}, (n' - 1, k' - 1, \rightarrow))$ in T , suppose that at the moment the item $(\text{rec}, (n' - 1, k', \leftarrow))$ (this is the left child) has been processed the current vertex x satisfies $x_{[1, n'-1]} = b_{n'-1, k'}$. Then at the moment the right child $(\text{rec}, (n' - 1, k' - 1, \rightarrow))$ is popped from the stack the current vertex x satisfies $x_{[1, n'-1]} = b_{n'-1, k'-1}$.
- (6) Given a node $(\text{rec}, (n', k', \text{dir}))$, $\text{dir} \in \{\leftarrow, \rightarrow\}$, in T , suppose that at the moment that this item is popped from the stack the current vertex x satisfies $x_{[1, n']} = a_{n', k'}$ if $\text{dir} = \leftarrow$ or $x_{[1, n']} = b_{n', k'}$ if $\text{dir} = \rightarrow$. Moreover, if this node is not a leaf, then suppose both children satisfy properties (i) or (ii) of the lemma. Then, this item satisfies property (i) or (ii) of the lemma (if $\text{dir} = \leftarrow$ or $\text{dir} = \rightarrow$, respectively).

Note that all these auxiliary statements except (1) are conditional. However, the unconditional properties (i) and (ii) of the lemma can be easily derived from the conditional statements (1)–(6)

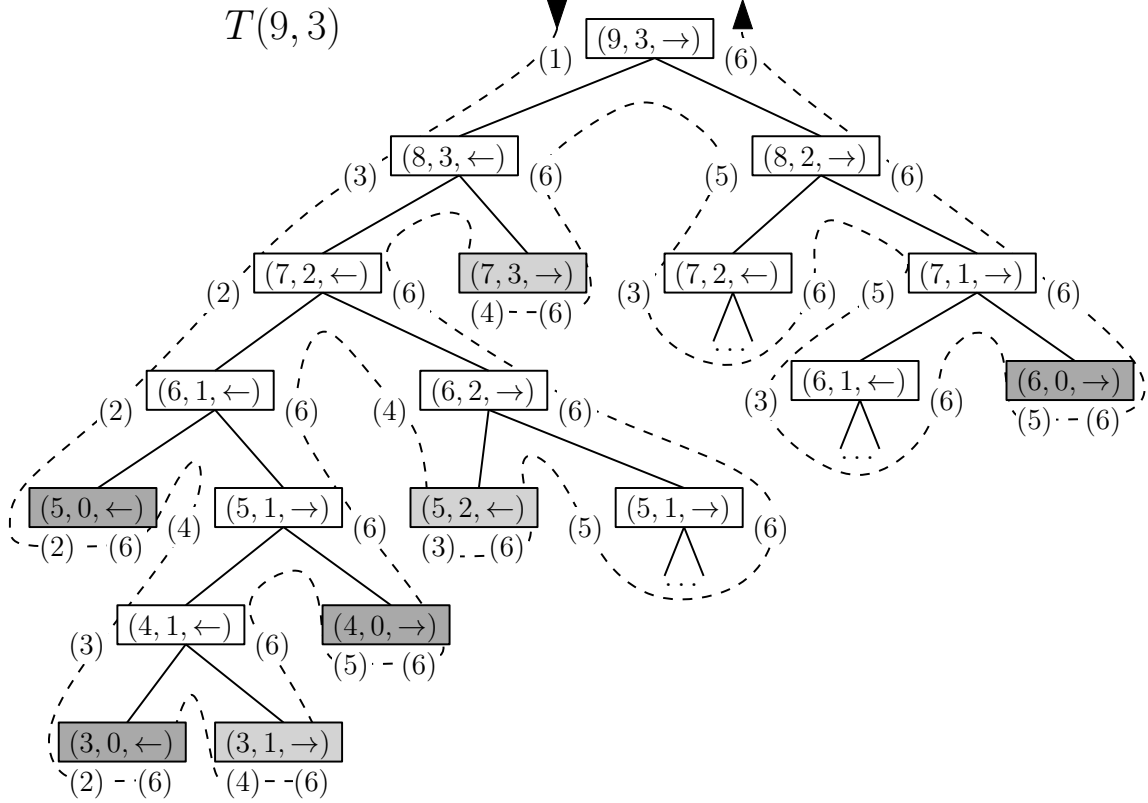


FIGURE 6. The recursion tree $T = T(n, k)$ for $n = 9$ and $k = 3$ and other notations used in the proof of Lemma 19. In the figure, the nodes $(\text{rec}, (n', k', \text{dir}))$ of T are represented by the triples (n', k', dir) . The dashed line indicates a left-to-right tree traversal order in which the conditional auxiliary statements (1)–(6) are applied to derive the unconditional properties (i) and (ii) of the lemma.

by applying them to all nodes of the recursion tree T along a left-to-right tree traversal, starting by applying (1) to the root of T , which is the item $(\text{rec}, (n, k, \rightarrow))$ (see the dashed line in Figure 6).

We proceed to prove the auxiliary statements mentioned before.

- (1) This follows easily from the instructions in lines S1–S5. Specifically, the item $(\text{rec}, (n, k, \rightarrow))$ is popped from the stack in the third iteration of the while-loop, after the bits at positions $n - k$ and n have been flipped, leading from the initial vertex $a_{n,k}$ via $a_{n,k+1}$ to the vertex $x = b_{n,k}$.
- (2) When the node $(\text{rec}, (n', k', \leftarrow))$ is popped from the stack, it is replaced by four new items, and the left child $(\text{rec}, (n' - 1, k' - 1, \leftarrow))$ becomes the new topmost item on the stack (see lines S21–S24). Therefore, this item will be popped from the stack in the next iteration of the while-loop, i.e., the value of x is not modified in between. From the assumption $x_{[1,n']} = a_{n',k'}$ and the definition (18) we obtain that $x_{[1,n'-1]} = a_{n'-1,k'-1}$.
- (3) When the node $(\text{rec}, (n', k', \rightarrow))$ is popped from the stack, it is replaced by four new items, and the left child $(\text{rec}, (n' - 1, k', \leftarrow))$ becomes the new topmost item on the stack (see lines S26–S29). Therefore, this item will be popped from the stack in the next iteration of the while-loop, i.e., the value of x is not modified in between. From the assumption $x_{[1,n']} = b_{n',k'}$ and the definition (18) we obtain that $x_{[1,n'-1]} = a_{n'-1,k'}$.
- (4) When the node $(\text{rec}, (n', k', \leftarrow))$ is popped from the stack, it is replaced by four new items, where the left child $(\text{rec}, (n' - 1, k' - 1, \leftarrow))$ becomes the new topmost item on the stack and the right child $(\text{rec}, (n' - 1, k', \rightarrow))$ is located three items below (see lines S21–S24). So when the

left child has been processed, then the algorithm runs two iterations where the bits at positions $n' - k' - 1$ (line S23) and n' (line S22) are flipped, and in the next iteration the right child ($\mathbf{rec}, (n' - 1, k', \rightarrow)$) is popped from the stack. From the assumption that before these two bitflips we have $x_{[1, n'-1]} = b_{n'-1, k'-1}$ and from the definition (18) we obtain that after these two bitflips the current vertex x satisfies $x_{[1, n'-1]} = b_{n'-1, k'}$.

- (5) When the node ($\mathbf{rec}, (n', k', \rightarrow)$) is popped from the stack, it is replaced by four new items, where the left child ($\mathbf{rec}, (n' - 1, k', \leftarrow)$) becomes the new topmost item on the stack and the right child ($\mathbf{rec}, (n' - 1, k' - 1, \rightarrow)$) is located three items below (see lines S26–S29). So when the left child has been processed, then the algorithm runs two iterations where the bits at positions n' (line S28) and $n' - k' - 1$ (line S27) are flipped, and in the next iteration the right child ($\mathbf{rec}, (n' - 1, k' - 1, \rightarrow)$) is popped from the stack. From the assumption that before these two bitflips we have $x_{[1, n'-1]} = b_{n'-1, k'}$ and from the definition (18) we obtain that after these two bitflips the current vertex x satisfies $x_{[1, n'-1]} = b_{n'-1, k'-1}$.
- (6) We distinguish three cases (by the conditions in lines S14 and S19, no other cases are possible):
 - (a) The node ($\mathbf{rec}, (n', k', \mathbf{dir})$) is a leaf and we have $k' = 0$ (the corresponding leafs of T are highlighted in darkgray in Figure 6). We consider the subcase that $\mathbf{dir} = \leftarrow$, and we need to prove that this item satisfies property (i) of the lemma. In this case, the item is simply popped from the stack (the condition on the value of x in this moment holds by assumption), and nothing is done, i.e., the item has been processed in the next iteration of the while-loop. In between, the variable x is not modified, so by the assumption $x_{[1, n']} = a_{n', 0} = 0^{n'}$ it also satisfies $x_{[1, n']} = b_{n', 0} = 0^{n'}$. Trivially, in between all vertices of the form $x' \circ x_{[n'+1, n]}$, $x' \in B_{n', 0} = \{0^{n'}\}$, except the vertex $a_{n', 0} \circ x_{[n'+1, n]} = 0^{n'} \circ x_{[n'+1, n]}$ (i.e., no vertices at all) are visited. Furthermore, neither the vertex $a_{n', 1} \circ x_{[n'+1, n]}$ nor the vertex $b_{n', 1} \circ x_{[n'+1, n]}$ are visited. This shows that the item ($\mathbf{rec}, (n', k', \mathbf{dir})$) indeed satisfies property (i) of the lemma. The other subcase that $\mathbf{dir} = \rightarrow$ can be handled analogously (in this case property (ii) also holds trivially).
 - (b) The node ($\mathbf{rec}, (n', k', \mathbf{dir})$) is a leaf and we have $k' \geq 1$ and $n' = 2k' + 1$ (the corresponding leafs of T are highlighted in lightgray in Figure 6). We consider the subcase that $\mathbf{dir} = \leftarrow$, and we need to prove that this item satisfies property (i) of the lemma. In this case, the item is popped from the stack (the condition on the value of x in this moment holds by assumption), and the algorithm issues the call $\mathbf{HAMCYCLE}(k', x, \leftarrow)$ in line S16. We know that upon termination of this algorithm, the current value of the variable x satisfies $x_{[1, n']} = b_{n', k'}$. Moreover, within the algorithm $\mathbf{HAMCYCLE}()$ all vertices of the form $x' \circ x_{[n'+1, n]}$, $x' \in B_{n', k'}$, except the vertex $a_{n', k'} \circ x_{[n'+1, n]}$ are visited. Furthermore, the vertex $a_{n', k'+1} \circ x_{[n'+1, n]}$ is not visited. This shows that the item ($\mathbf{rec}, (n', k', \mathbf{dir})$) indeed satisfies property (i) of the lemma. The other subcase that $\mathbf{dir} = \rightarrow$ can be handled analogously (in this case property (ii) holds by our definition of the algorithm $\mathbf{HAMCYCLE}(k', x, \rightarrow)$ called in line S18).
 - (c) The node ($\mathbf{rec}, (n', k', \mathbf{dir})$) is not a leaf, so we have $k' \geq 1$ and $n' > 2k' + 1$ (the corresponding inner nodes of T are drawn in white in Figure 6). We consider the subcase that $\mathbf{dir} = \leftarrow$, and we need to prove that this item satisfies property (i) of the lemma. In this case, the item is popped from the stack (the condition on the value of x in this moment holds by assumption) and replaced by four new items ($\mathbf{rec}, (n' - 1, k', \rightarrow)$), (\mathbf{flip}, n'), ($\mathbf{flip}, n' - k' - 1$) and ($\mathbf{rec}, (n' - 1, k' - 1, \leftarrow)$), pushed onto the stack in this order in lines S21–S24. By the assumptions that the left child ($\mathbf{rec}, (n' - 1, k' - 1, \leftarrow)$) of the node ($\mathbf{rec}, (n', k', \mathbf{dir})$) in T satisfies property (i) of the lemma and the right child ($\mathbf{rec}, (n' - 1, k', \rightarrow)$) satisfies property (ii) of the lemma, we obtain the following: When the item ($\mathbf{rec}, (n', k', \mathbf{dir})$) is popped from the stack (and replaced by the four new items mentioned above), by assumption we know that the current vertex x satisfies $x = a_{n', k'} \circ$

$x_{[n'+1,n]} = a_{n'-1,k'-1} \circ 1 \circ x_{[n'+1,n]}$. Until the left child ($\text{rec}, (n' - 1, k' - 1, \leftarrow)$) has been processed, the algorithm visits all vertices of the form $x' \circ 1 \circ x_{[n'+1,n]}$, $x' \in B_{n'-1,k'-1}$, except the vertex $a_{n'-1,k'-1} \circ 1 \circ x_{[n'+1,n]} = a_{n',k'} \circ x_{[n'+1,n]}$, no vertex twice. Furthermore, neither the vertex $a_{n'-1,k'} \circ 1 \circ x_{[n'+1,n]} = a_{n',k'+1} \circ x_{[n'+1,n]}$ nor the vertex $b_{n'-1,k'} \circ 1 \circ x_{[n'+1,n]}$ are visited. At this point, the current vertex x satisfies $x = b_{n'-1,k'-1} \circ 1 \circ x_{[n'+1,n]}$. After that, the **flip**-item ($\text{flip}, n' - k' - 1$) is popped from the stack, so in the next iteration of the while-loop the algorithm visits the vertex $x = b_{n'-1,k'} \circ 1 \circ x_{[n'+1,n]}$ (for the first time). After that, the **flip**-item (flip, n') is popped from the stack, so in the next iteration of the while-loop the algorithm visits the vertex $x = b_{n'-1,k'} \circ 0 \circ x_{[n'+1,n]}$ (for the first time). Until the right child ($\text{rec}, (n' - 1, k', \rightarrow)$) has been processed, the algorithm visits all vertices of the form $x' \circ 0 \circ x_{[n'+1,n]}$, $x' \in B_{n'-1,k'}$, except the vertex $b_{n'-1,k'} \circ 0 \circ x_{[n'+1,n]}$ (this vertex has been visited just before), no vertex twice. Furthermore, neither the vertex $a_{n'-1,k'+1} \circ 0 \circ x_{[n'+1,n]} = b_{n',k'+1} \circ x_{[n'+1,n]}$ nor the vertex $b_{n'-1,k'+1} \circ 0 \circ x_{[n'+1,n]}$ are visited. At this point, the current vertex x satisfies $x = a_{n'-1,k'} \circ 0 \circ x_{[n'+1,n]} = b_{n',k'} \circ x_{[n',n]}$. Summarizing, the item ($\text{rec}, (n', k', \text{dir})$) indeed satisfies property (i) in the lemma. The other subcase that $\text{dir} = \rightarrow$ can be handled analogously, considering which four items are pushed onto the stack in lines S26–S29 after the item ($\text{rec}, (n', k', \text{dir})$) is popped.

This completes the proof of the lemma. \square

From Lemma 19 we conclude that the algorithm $\text{SATCYCLE}(n, k)$ indeed computes a saturating cycle in $Q_{n,[k,k+1]}$: By the instructions in lines S1–S5, we start at the vertex $x = a_{n,k}$, and in the first two iterations of the while-loop, two **flip**-items are popped from the stack (in the reverse order in which they are pushed onto the stack), which entail the calls $\text{VISIT}(a_{n,k+1})$ and $\text{VISIT}(b_{n,k})$. In the next iteration, the **rec**-item ($\text{rec}, (n, k, \rightarrow)$) is popped from the stack, so by property (ii) from Lemma 19, from this point in time until this item has been processed (i.e., until the algorithm terminates), the algorithm visits a saturating path in $Q_{n,[k,k+1]}$ that ends at the vertex $a_{n,k}$ (this is the last visited vertex). Together with the first two visited vertices this saturating path forms a saturating cycle in the graph $Q_{n,[k,k+1]}$.

4.3.2. Running time and space requirements of the algorithm. The running time of the algorithms $\text{HAMCYCLE}(k, x, \text{dir})$, $\text{dir} \in \{\leftarrow, \rightarrow\}$, called in lines S16 and S18 is $\mathcal{O}(1)$ on average per visited vertex plus $\mathcal{O}(k)$ for the initialization (as mentioned in [MN16], the initialization time is only $\mathcal{O}(k)$ instead of the general bound $\mathcal{O}(k^2)$ when the starting vertex is prescribed, as $a_{2k+1,k}$ or $b_{2k+1,k}$ in our case). Since in total $2^{\binom{2k+1}{k}} - 2 = 2^{\Theta(k)}$ vertices are visited in each call, the $\mathcal{O}(k)$ initialization time can be discounted to the $\mathcal{O}(1)$ average time per visited vertex.

The next lemma is needed to bound the running time of the remaining operations of the algorithm $\text{SATCYCLE}()$.

Lemma 20. *The algorithm $\text{SATCYCLE}(n, k)$ satisfies the following invariant for all inputs $n \geq 3$ and $1 \leq k \leq \lfloor (n-1)/2 \rfloor$:*

- (i) *Any two consecutive **rec**-items on the stack S are separated by exactly two **flip**-items. At the bottom of the stack S is a single **rec**-item, and on the top of the stack are at most two **flip**-items.*
- (ii) *Consider all **rec**-items $(\text{rec}, (n_1, k_1, \text{dir}_1)), \dots, (\text{rec}, (n_r, k_r, \text{dir}_r))$, on the stack S from bottom to top. Then we have $n_1 > n_2 > \dots > n_{r-1} \geq n_r \geq 3$.*

Proof. These properties follow immediately from the instructions in lines S3–S5, S13, S21–S24, S26–S29. \square

From Lemma 20 (ii) we conclude that the stack S has height at most $3n$.

Consider the set R of all **rec**-items and the set F of all **flip**-items that appear on the stack S in the course of the algorithm ($R \cup F$ is the node set of the tree $T(n, k)$ defined in the previous section). We define an injection $R \rightarrow F$ as follows (see Lemma 20 (i)): The **rec**-item pushed onto the stack in line S3 is mapped onto the **flip**-item pushed onto the stack in line S4. Moreover, the **rec**-items pushed in lines S21 and S24 are mapped onto the **flip**-items pushed in lines S22 and S23, respectively. Similarly, the **rec**-items pushed in lines S26 and S29 are mapped onto the **flip**-items pushed in lines S27 and S28, respectively. From this injection it follows that $|R| \leq |F|$. Note that processing each stack item takes only constant time, where by processing we mean popping it from the stack and either issuing the call to `HAMCYCLE()` (without the time spent inside this function) or pushing four other items onto the stack. Combining this with the bound $|F| \geq |R|$ and using that each **flip**-item leads to a call `VISIT(x)` in line S11 yields that the algorithm spends on average $\mathcal{O}(1)$ time per visited vertex (outside of `HAMCYCLE()`). As we argued before, also the algorithm `HAMCYCLE()` needs only time $\mathcal{O}(1)$ on average to visit each vertex, so overall each vertex is generated in time $\mathcal{O}(1)$ on average.

The space required by our algorithm is only $\mathcal{O}(n)$, as the array x and the stack S have at most $\mathcal{O}(n)$ entries.

4.3.3. Proof of part (b) of Theorem 9. First of all the algorithm `SATCYCLE(n, k)` can easily be generalized to allow parameters $\lfloor n/2 \rfloor \leq k \leq n - 2$ by starting at the vertex $x := 1^{k+1}0^{n-k-1}$ and by applying the bitflip sequence computed by the original algorithm `SATCYCLE($n, n - k - 1$)` (which starts at the complement of x , the vertex $a_{n, n-k-1}$). Moreover, by mimicking the gluing approach from the proof of Theorem 7 (iii), the algorithm `SATCYCLE(n, k)` can be easily generalized to an algorithm `SATCYCLE(n, k, l)` that generates a saturating cycle in $Q_{n, [k, l]}$ for any even number of consecutive levels $[k, l]$ that are entirely below or above the middle (suitably string together multiple calls `SATCYCLE(n, k')`, $k \leq k' \leq l$; for details, see our C++ implementation). All the previously established bounds for the running time and space requirements carry over to this setting.

To obtain analogous algorithms for tight enumerations, one first derives an algorithm `TIGHTENUM(n, k)` that computes a tight enumeration of the vertices of $Q_{n, [k, k+1]}$ (this algorithm mimicks the proof of Theorem 8 (iia) and is very much analogous to the algorithm `SATCYCLE(n, k)`, using a stack). Mimicking the gluing approach from the proof of Theorem 8 (iib), this algorithm can then be generalized to an algorithm `TIGHTENUM(n, k, l)` that generates a tight enumeration of the vertices of $Q_{n, [k, l]}$ for any even number of levels $[k, l]$ that are entirely below or above the middle (for details, see our C++ implementation).

This proves part (b) of Theorem 9.

5. PROOF OF THEOREM 10

We now present the proof of Theorem 10.

Proof of Theorem 10. For $c = 0$ the claim follows from Theorem 1, so we can assume that $c \geq 1$.

The cycle obtained from Theorem 11 by trimming the reflected Gray code to levels $[k - c, k + 1 + c]$ of Q_{2k+1} misses exactly

$$m := 2 \left(\binom{2k+1}{k+c+1} - u_{2k+1, k+c} \right) \stackrel{(21)}{=} 2 \left(\binom{2k+1}{k+c+1} - \binom{2k}{k+c} \right) = 2 \binom{2k}{k+c+1} \quad (23)$$

many vertices. The total number of vertices of $Q_{2k+1,[k-c,k+1+c]}$ satisfies the relation

$$v := 2 \left(\binom{2k+1}{k+c+1} + \binom{2k+1}{k+c} + \cdots + \binom{2k+1}{k+1} \right) \geq 2(c+1) \binom{2k+1}{k+c+1} \geq 4(c+1) \binom{2k}{k+c+1} \quad (24)$$

(recall (17a)). Combining (23) and (24) shows that

$$\frac{m}{v} \leq \frac{1}{2(c+1)} \quad , \quad (25)$$

yielding the first bound claimed in Theorem 10.

To derive the second bound, we consider another way of building a long cycle in $Q_{2k+1,[k-c,k+1+c]}$. For odd $c \geq 1$ we glue together $c+1$ many saturating cycles (obtained from Theorem 3) between the pairs of consecutive levels $(k-c, k-c+1), \dots, (k-1, k), (k+1, k+2), \dots, (k+c, k+c+1)$ as in the proof of Theorem 7 (iii) (see Figure 5). The number of missed vertices m in this case satisfies the relation

$$\begin{aligned} m &= 2 \left(\binom{2k+1}{k+1} - \binom{2k+1}{k+2} + \binom{2k+1}{k+3} - \binom{2k+1}{k+4} + \cdots + \binom{2k+1}{k+c} - \binom{2k+1}{k+c+1} \right) \\ &= 2 \left(\binom{2k}{k} - \binom{2k}{k+c+1} \right) \leq 2 \left(\left(\frac{k+1}{k-c} \right)^{c+1} - 1 \right) \binom{2k}{k+c+1} . \end{aligned} \quad (26)$$

Combining (24) and (26) and using that $1 - x \leq \exp(-x)$ we obtain

$$\frac{m}{v} \leq \frac{1}{2(c+1)} \left(\left(\frac{k+1}{k-c} \right)^{c+1} - 1 \right) \leq \frac{1}{2(c+1)} \left(\exp \left(\frac{(c+1)^2}{k-c} \right) - 1 \right) . \quad (27)$$

For even $c \geq 2$ we glue together $c+1$ many saturating cycles between the pairs of consecutive levels $(k-c, k-c+1), \dots, (k, k+1), \dots, (k+c, k+c+1)$ as before. In this case the number of missed vertices m can be computed very similarly to before as $2(\binom{2k}{k+1} - \binom{2k}{k+c+1})$, and this expression is bounded from above by the expressions in (26), implying that the same bound (27) holds also in this case.

Combining the bounds (25) and (27) completes the proof of the theorem. \square

ACKNOWLEDGEMENTS

The authors thank Jiří Fink, Jerri Nummenpalo and Robert Šámal for several stimulating discussions about the problems discussed in this paper. The first author acknowledges the support by the Czech Science Foundation grant GA14-10799S.

REFERENCES

- [BER76] J. Bitner, G. Ehrlich, and E. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Comm. ACM*, 19(9):517–521, 1976.
- [BW84] M. Buck and D. Wiedemann. Gray codes with restricted density. *Discrete Math.*, 48(2-3):163–171, 1984.
- [Cha89] P. Chase. Combination generation and graylex ordering. *Congr. Numer.*, 69:215–242, 1989. Eighteenth Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, MB, 1988).
- [DG12] P. Diaconis and R. Graham. *Magical mathematics*. Princeton University Press, Princeton, NJ, 2012. The mathematical ideas that animate great magic tricks, With a foreword by Martin Gardner.
- [DKS94] D. Duffus, H. Kierstead, and H. Snevily. An explicit 1-factorization in the middle of the Boolean lattice. *J. Combin. Theory Ser. A*, 65(2):334–342, 1994.
- [DSW88] D. Duffus, B. Sands, and R. Woodrow. Lexicographic matchings cannot form Hamiltonian cycles. *Order*, 5(2):149–161, 1988.
- [EHH01] M. El-Hashash and A. Hassan. On the Hamiltonicity of two subgraphs of the hypercube. In *Proceedings of the Thirty-second Southeastern International Conference on Combinatorics, Graph Theory and Computing (Baton Rouge, LA, 2001)*, volume 148, pages 7–32, 2001.

- [Ehr73] G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. Assoc. Comput. Mach.*, 20:500–513, 1973.
- [EHR84] P. Eades, M. Hickey, and R. Read. Some Hamilton paths and a minimal change algorithm. *J. Assoc. Comput. Mach.*, 31(1):19–29, 1984.
- [EM84] P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Inform. Process. Lett.*, 19(3):131–133, 1984.
- [FT95] S. Felsner and W. Trotter. Colorings of diagrams of interval orders and α -sequences of sets. *Discrete Math.*, 144(1-3):23–31, 1995. Combinatorics of ordered sets (Oberwolfach, 1991).
- [Gra] F. Gray. Pulse code communication. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.
- [GŠ10] P. Gregor and R. Škrekovski. On generalized middle-level problem. *Inform. Sci.*, 180(12):2448–2457, 2010.
- [Hav83] I. Havel. Semipaths in directed cubes. In *Graphs and other combinatorial topics (Prague, 1982)*, volume 59 of *Teubner-Texte Math.*, pages 101–108. Teubner, Leipzig, 1983.
- [HKRR05] P. Horák, T. Kaiser, M. Rosenfeld, and Z. Ryjáček. The prism over the middle-levels graph is Hamiltonian. *Order*, 22(1):73–81, 2005.
- [JM95] T. Jenkyns and D. McCarthy. Generating all k -subsets of $\{1 \dots n\}$ with minimal changes. *Ars Combin.*, 40:153–159, 1995.
- [Joh04] R. Johnson. Long cycles in the middle two layers of the discrete cube. *J. Combin. Theory Ser. A*, 105(2):255–271, 2004.
- [Knu11] D. Knuth. *The Art of Computer Programming, Volume 4A*. Addison-Wesley, 2011.
- [KT88] H. Kierstead and W. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988.
- [LS03] S. Locke and R. Stong. Problem 10892: Spanning cycles in hypercubes. *Amer. Math. Monthly*, 110:440–441, 2003.
- [MN15] T. Mütze and J. Nummenpalo. Efficient computation of middle levels Gray codes. In N. Bansal and I. Finocchi, editors, *Algorithms - ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 915–927. Springer Berlin Heidelberg, 2015. *arXiv:1506.07898*.
- [MN16] T. Mütze and J. Nummenpalo. A constant-time algorithm for middle levels Gray codes. *arXiv:1606.06172*. An extended abstract has been accepted for presentation at SODA 2017, June 2016.
- [MS15] T. Mütze and P. Su. Bipartite Kneser graphs are Hamiltonian. In J. Nešetřil, O. Serra, and J.A. Telle, editors, *Electronic Notes in Discrete Mathematics - Eurocomb 2015*, volume 49, pages 259–267. Elsevier, 2015. *arXiv:1503.09175*. Accepted for publication in *Combinatorica*.
- [Müt16] T. Mütze. Proof of the middle levels conjecture. *Proc. Lond. Math. Soc.*, 112(4):677–713, 2016.
- [MW12] T. Mütze and F. Weber. Construction of 2-factors in the middle layer of the discrete cube. *J. Combin. Theory Ser. A*, 119(8):1832–1855, 2012.
- [Rus88] F. Ruskey. Adjacent interchange generation of combinations. *J. Algorithms*, 9(2):162–180, 1988.
- [SA11] M. Shimada and K. Amano. A note on the middle levels conjecture. *arXiv:0912.4564*, Sep 2011.
- [Sav93] C. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35-A:97–108, 1993.
- [Sav97] C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997.
- [SSS09] I. Shields, B. Shields, and C. Savage. An update on the middle levels problem. *Discrete Math.*, 309(17):5271–5277, 2009.
- [SW95] C. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995.
- [SW14] B. Stevens and A. Williams. The coolest way to generate binary strings. *Theory Comput. Syst.*, 54(4):551–577, 2014.
- [TL73] D. Tang and C. Liu. Distance-2 cyclic chaining of constant-weight codes. *IEEE Trans. Computers*, C-22:176–180, 1973.
- [Win04] P. Winkler. *Mathematical puzzles: a connoisseur's collection*. A K Peters, Ltd., Natick, MA, 2004.
- [www] currently <http://www.math.tu-berlin.de/~muetze>.